

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Alfred Kobsa

University of California, Irvine, CA, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

University of Dortmund, Germany

Madhu Sudan

Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Gerhard Weikum

Max-Planck Institute of Computer Science, Saarbruecken, Germany

Xiaodong Li Michael Kirley
Mengjie Zhang David Green
Vic Ciesielski Hussein Abbass
Zbigniew Michalewicz Tim Hendtlass
Kalyanmoy Deb Kay Chen Tan
Jürgen Branke Yuhui Shi (Eds.)

Simulated Evolution and Learning

7th International Conference, SEAL 2008
Melbourne, Australia, December 7-10, 2008
Proceedings

Volume Editors

Xiaodong Li, RMIT University, Melbourne, Australia, xiaodong.li@rmit.edu.au

Michael Kirley, The University of Melbourne, Australia, mkirley@cs.mu.oz.au

Mengjie Zhang

Victoria University of Wellington, New Zealand, mengjie.zhang@mcs.vuw.ac.nz

David Green, Monash University, Australia, David.Green@infotech.monash.edu.au

Vic Ciesielski, RMIT University, Melbourne, Australia, vc@rmit.edu.au

Hussein Abbass

University of New South Wales, Canberra, Australia, h.abbass@adfa.edu.au

Zbigniew Michalewicz, University of Adelaide, Australia, zbyszek@cs.adelaide.edu.au

Tim Hendtlass

Swinburne University of Technology, Melbourne, Australia, thendtlass@swin.edu.au

Kalyanmoy Deb

Indian Institute of Technology Kanpur, Uttar Pradesh, India, deb@iitk.ac.in

Kay Chen Tan, National University of Singapore, eletankc@nus.edu.sg

Jürgen Branke, University of Karlsruhe, Germany, branke@aifb.uni-karlsruhe.de

Yuhui Shi

Xi'an Jiaotong-Liverpool University, Suzhou, China, yuhui.shi@xjtlu.edu.cn

Library of Congress Control Number: 2008939923

CR Subject Classification (1998): F.1.1, I.2.6, I.6, G.1.6, D.2.2, J.3-4

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743

ISBN-10 3-540-89693-7 Springer Berlin Heidelberg New York

ISBN-13 978-3-540-89693-7 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

springer.com

© Springer-Verlag Berlin Heidelberg 2008

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper SPIN: 12573189 06/3180 5 4 3 2 1 0

Preface

This LNCS volume contains the papers presented at SEAL 2008, the 7th International Conference on Simulated Evolution and Learning, held December 7–10, 2008, in Melbourne, Australia. SEAL is a prestigious international conference series in evolutionary computation and learning. This biennial event was first held in Seoul, Korea, in 1996, and then in Canberra, Australia (1998), Nagoya, Japan (2000), Singapore (2002), Busan, Korea (2004), and Hefei, China (2006).

SEAL 2008 received 140 paper submissions from more than 30 countries. After a rigorous peer-review process involving at least 3 reviews for each paper (i.e., over 420 reviews in total), the best 65 papers were selected to be presented at the conference and included in this volume, resulting in an acceptance rate of about 46%.

The papers included in this volume cover a wide range of topics in simulated evolution and learning: from evolutionary learning to evolutionary optimization, from hybrid systems to adaptive systems, from theoretical issues to real-world applications. They represent some of the latest and best research in simulated evolution and learning in the world.

The conference featured four distinguished keynote speakers: Kalyanmoy Deb, Zbigniew Michalewicz, Xin Yao and Hussein Abbass. Kalyanmoy Deb's talk was on "Reliability-Based Optimization for Handling Uncertainty in Evolutionary Algorithms." Zbigniew Michalewicz's talk was on "The Future of Business Intelligence." Xin Yao's talk was on "Cooperative Coevolution for Large-Scale Evolutionary Optimization," while Hussein Abbass's talk was on "The Future of Intelligent Systems Is Non-Dominance." We were very fortunate to have such internationally renown research leaders giving talks at SEAL 2008, given their busy schedules. Their presence at the conference was yet another indicator of the importance of the SEAL conference series on the international research map.

SEAL 2008 also included five tutorials, which were free to all conference participants. Four tutorials were kindly provided by the four keynote speakers, and in addition, we were also fortunate to have Dipankar Dasgupta present a tutorial of a fascinating topic on "Immunological Computation." These five tutorials covered some of the hottest topics in evolutionary computation and learning and their applications, including "Recent Trends in Evolutionary Multi-Objective Optimization" (Kalyanmoy Deb), "Puzzle-Based Learning" (Zbigniew Michalewicz), "Evolving and Designing Neural Network Ensembles" (Xin Yao), "Getting Evolution to Solve Your Practical Problems" (Hussein Abbass), and "Immunological Computation" (Dipankar Dasgupta). They provided an excellent start to the four-day conference.

The success of a conference depends on its authors, reviewers and organizers. SEAL 2008 was no exception. We are very grateful to all the authors for their paper submissions and to all the Program Committee members for their outstanding

effort in reviewing the papers within a very tight schedule. We relied heavily upon a team of volunteers to keep SEAL 2008 running smoothly and efficiently. They were the true heroes working behind the scene. In particular, Donna-Lee Stanes and Dianne Nguyen from Monash University played an important role in supporting the running of the conference. We are most grateful to all the volunteers for their great efforts and contributions.

We would also like to thank our sponsor for providing all the support and financial assistance, including the ECML group, RMIT University; the Department of Computer Science and Software Engineering, University of Melbourne; School of Information Technology, Monash University; ARC Complex Open Systems Research Network; and the Air Force Office of Scientific Research, Asian Office of Aerospace Research and Development (AFOSR/AOARD).

September 2008

Xiaodong Li
Michael Kirley
Mengjie Zhang
David Green

Organization

The 7th International Conference on Simulated Evolution And Learning (SEAL 2008) was organized and hosted by RMIT University, The University of Melbourne, and Monash University, Melbourne, Australia.

SEAL 2008 Conference Committee

General Chair	Xiaodong Li (Australia)
Program Chairs	Michael Kirley (Australia) Mengjie Zhang (New Zealand) David Green (Australia)
Technical Co-chairs	Vic Ciesielski (Australia) Hussein Abbass (Australia) Zbigniew Michalewicz (Australia) Tim Hendtlass (Australia) Kalyanmoy Deb (India) Kay Chen Tan (Singapore) Jürgen Branke (Germany) Yuhui Shi (China)
Tutorials and Special Sessions Chair	Cara MacNish (Australia)
Organizing Committee Co-chairs	Andy Song (Australia) Dianne Nguyen (Australia)
Organizing Committee Members	Upali K. Wickramasinghe, Stefan Bird, Gayan Wijesinghe, Antony Iorio, and Golriz Rezaei

International Advisory Committee

Takeshi Furuhashi	Japan
Jong-Hwan Kim	South Korea
Bob McKay	South Korea
Lipo Wang	Singapore
Xin Yao	UK

SEAL 2008 Tutorials

Recent Trends in Evolutionary Multi-Objective Optimization

Kalyanmoy Deb

Puzzle-Based Learning

Zbigniew Michalewicz

Evolving and Designing Neural Network Ensembles

Xin Yao

Getting Evolution to Solve Your Practical Problems

Hussein Abbass

Immunological Computation

Dipankar Dasgupta

Sponsoring Institutions

Evolutionary Computation and Machine Learning Group (ECML), School of
Computer Science and IT, RMIT University

Department of Computer Science and Software Engineering, The University of
Melbourne

School of Information Technology, Monash University

ARC Complex Open Systems Research Network

Air Force Office of Scientific Research, Asian Office of Aerospace Research and
Development (AFOSR/AOARD)

Acknowledgements

We wish to thank the Air Force Office of Scientific Research, Asian Office of
Aerospace Research and Development (AFOSR/AOARD), for their contribu-
tion to the success of this conference.

Disclaimer: AFOSR/AOARD support is not intended to express or imply en-
dorsement by the U.S. Federal Government.

Program Committee

Hussein Abbass	David Jackson	Jani Rönkkönen
Hernan Aguirre	Licheng Jiao	Ramón Sagarna
Dan Angus	Mark Johnston	Sancho Salcedo-Sanz
Luigi Barone	Gul Muhammad Khan	Muhammad Sarfraz
Bir Bhanu	Michael Kirley	Ruhul Sarker
Stefan Bird	Tomas Klos	Zhongzhi Shi
Jürgen Branke	Mario Koeppen	Yuhui Shi
Lam Bui	Krzysztof Krawiec	Josefina Sierra
Stefano Cagnoni	Saku Kukkonen	Hiroshi Someya
Jinhai Cai	Robert K. Lai	Qing Song
Zhenjiang Cai	Per Kristian Lehre	Andy Song
Ying-ping Chen	Jinyan Li	Kay Chen Tan
Yi-Yuan Chiang	Jin Li	Ke Tang
Raymond Chiong	Wei Li	Krzysztof Trojanowski
Siang Yew Chong	Bin Li	Peter Vamplew
Vic Ciesielski	Xiaodong Li	Yu-Xuan Wang
Maurice Clerc	Jing Liu	Dianhui Wang
Peter Cowling	Wenjian Luo	Peter Whigham
Kalyanmoy Deb	Cara MacNish	Upali K.
Hepu Deng	Jon McCormack	Wickramasinghe
Grant Dick	Kathryn Merrick	Clinton Woodward
Marc Ebner	Bernd Meyer	Jason Xie
Daryl Essam	Ammar Mohemmed	Feng Xue
David Green	Irene Moser	Shengxiang Yang
Gary Greenfield	Kourosh Neshatian	Tina Yu
Steven Gustafson	David Newth	Lean Yu
Hisashi Handa	Gustavo Olague	Mengjie Zhang
Jingsong He	Yew Soon Ong	Shichao Zhang
Tim Hendtlass	Angel Perez-Bellido	Qingfu Zhang
Samuelson W. Hong	Lukas Pichl	Jun Zhang
Daniel Howard	Kai Qin	Shude Zhou
Antony Iorio	Han Yang Quek	Zhihua Zhou
Hisao Ishibuchi	Marcus Randall	

Table of Contents

Evolutionary Learning

Modelling Behaviour Cycles for Life-Long Learning in Motivated Agents	1
<i>Kathryn Merrick</i>	
Breaking the Synaptic Dogma: Evolving a Neuro-inspired Developmental Network.....	11
<i>Gul Muhammad Khan, Julian F. Miller, and David M. Halliday</i>	
A New Approach to Adapting Control Parameters in Differential Evolution Algorithm	21
<i>Liang Feng, Yin-Fei Yang, and Yu-Xuan Wang</i>	
A Novel Genetic Algorithm with Orthogonal Prediction for Global Numerical Optimization	31
<i>Jun Zhang, Jing-Hui Zhong, and Xiao-Min Hu</i>	
Phylogeny Inference Using a Multi-objective Evolutionary Algorithm with Indirect Representation	41
<i>Md. Rafiul Hassan, M. Maruf Hossain, C.K. Karmakar, and Michael Kirley</i>	
Evolved Look-Up Tables for Simulated DNA Controlled Robots.....	51
<i>Gary Greenfield</i>	
Multi-objective Improvement of Software Using Co-evolution and Smart Seeding	61
<i>Andrea Arcuri, David Robert White, John Clark, and Xin Yao</i>	
Policy Evolution with Grammatical Evolution	71
<i>Yow Tzu Lim, Pau Chen Cheng, John Andrew Clark, and Pankaj Rohatgi</i>	
A PSO Based Adaboost Approach to Object Detection	81
<i>Ammar W. Mohemmed, Mengjie Zhang, and Mark Johnston</i>	
Adaptive Non-uniform Distribution of Quantum Particles in mQSO	91
<i>Krzysztof Trojanowski</i>	
Genetically Evolved Fuzzy Rule-Based Classifiers and Application to Automotive Classification	101
<i>Teck Wee Chua and Woei Wan Tan</i>	
Improving XCS Performance by Distribution	111
<i>Urban Richter, Holger Prothmann, and Hartmut Schmeck</i>	

Evolving an Ensemble of Neural Networks Using Artificial Immune Systems	121
<i>Bruno H.G. Barbosa, Lam T. Bui, Hussein A. Abbass, Luis A. Aguirre, and Antônio P. Braga</i>	
Improving the Performance and Scalability of Differential Evolution	131
<i>Antony W. Iorio and Xiaodong Li</i>	
A Fuzzy-GA Decision Support System for Enhancing Postponement Strategies in Supply Chain Management	141
<i>Cassandra X.H. Tang and Henry C.W. Lau</i>	

Evolutionary Optimisation

Solving the Delay-Constrained Capacitated Minimum Spanning Tree Problem Using a Dandelion-Encoded Evolutionary Algorithm.....	151
<i>Ángel M. Pérez-Bellido, Sancho Salcedo-Sanz, Emilio G. Ortiz-García, Antonio Portilla-Figueras, and Maurizio Naldi</i>	
Generalized Extremal Optimization for Solving Multiprocessor Task Scheduling Problem	161
<i>Piotr Switalski and Franciszek Seredynski</i>	
Improving NSGA-II Algorithm Based on Minimum Spanning Tree	170
<i>Miqing Li, Jinhua Zheng, and Jun Wu</i>	
An Island Based Hybrid Evolutionary Algorithm for Optimization	180
<i>Changhe Li and Shengxiang Yang</i>	
A Particle Swarm Optimization Based Algorithm for Fuzzy Bilevel Decision Making with Objective-Shared Followers	190
<i>Ya Gao, Guangquan Zhang, and Jie Lu</i>	
Reference Point-Based Particle Swarm Optimization Using a Steady-State Approach	200
<i>Richard Allmendinger, Xiaodong Li, and Jürgen Branke</i>	
Genetic Algorithm Based Methods for Identification of Health Risk Factors Aimed at Preventing Metabolic Syndrome	210
<i>Topon Kumar Paul, Ken Ueno, Koichiro Iwata, Toshio Hayashi, and Nobuyoshi Honda</i>	
Extremal Optimisation and Bin Packing	220
<i>Tim Hendtlass and Marcus Randall</i>	
Extremal Optimisation with a Penalty Approach for the Multidimensional Knapsack Problem	229
<i>Pedro Gómez-Meneses and Marcus Randall</i>	

A Generator for Multimodal Test Functions with Multiple Global Optima	239
<i>Jani Rönkkönen, Xiaodong Li, Ville Kyrki, and Jouni Lampinen</i>	
Choosing Leaders for Multi-objective PSO Algorithms Using Differential Evolution	249
<i>Upali Wickramasinghe and Xiaodong Li</i>	
Comparison between Genetic Algorithm and Genetic Programming Performance for Photomosaic Generation	259
<i>Shahrul Badariah Mat Sah, Vic Ciesielski, Daryl D'Souza, and Marsha Berry</i>	
Parameter Tuning of Real-Valued Crossover Operators for Statistics Preservation	269
<i>Hiroshi Someya</i>	
Hybrid Particle Swarm Optimization Based on Thermodynamic Mechanism	279
<i>Yu Wu, Yuanxiang Li, Xing Xu, and Sheng Peng</i>	
Multiagent Evolutionary Algorithm for T-coloring Problem	289
<i>Jing Liu, Weicai Zhong, and Jinshu Li</i>	
Non-photorealistic Rendering Using Genetic Programming	299
<i>Perry Barile, Vic Ciesielski, and Karen Trist</i>	
Use of Local Ranking in Cellular Genetic Algorithms with Two Neighborhood Structures	309
<i>Hisao Ishibuchi, Noritaka Tsukamoto, and Yusuke Nojima</i>	
Information Theoretic Classification of Problems for Metaheuristics	319
<i>Kent C.B. Steer, Andrew Wirth, and Saman K. Halgamuge</i>	
Task Decomposition for Optimization Problem Solving	329
<i>Ehab Z. Elfeky, Ruhul A. Sarker, and Daryl L. Essam</i>	
Discussion of Search Strategy for Multi-objective Genetic Algorithm with Consideration of Accuracy and Broadness of Pareto Optimal Solutions	339
<i>Tomoyuki Hiroyasu, Masashi Nishioka, Mitsunori Miki, and Hisatake Yokouchi</i>	
Discussion of Offspring Generation Method for Interactive Genetic Algorithms with Consideration of Multimodal Preference	349
<i>Fuyuko Ito, Tomoyuki Hiroyasu, Mitsunori Miki, and Hisatake Yokouchi</i>	

Solving Very Difficult Japanese Puzzles with a Hybrid Evolutionary-Logic Algorithm	360
<i>Emilio G. Ortiz-García, Sancho Salcedo-Sanz, Ángel M. Pérez-Bellido, Antonio Portilla-Figueras, and Xin Yao</i>	
Joint Multicast Routing and Channel Assignment in Multiradio Multichannel Wireless Mesh Networks Using Simulated Annealing	370
<i>Hui Cheng and Shengxiang Yang</i>	
General Game Playing with Ants	381
<i>Shiven Sharma, Ziad Kobti, and Scott Goodwin</i>	
A Generalized Approach to Construct Benchmark Problems for Dynamic Optimization	391
<i>Changhe Li and Shengxiang Yang</i>	
A Study on the Performance of Substitute Distance Based Approaches for Evolutionary Many Objective Optimization	401
<i>Hemant K. Singh, Amitay Isaacs, Tapabrata Ray, and Warren Smith</i>	
Performance Evaluation of an Adaptive Ant Colony Optimization Applied to Single Machine Scheduling	411
<i>Davide Anghinolfi, Antonio Boccalatte, Massimo Paolucci, and Christian Vecchiola</i>	
Robust Optimization by ϵ -Ranking on High Dimensional Objective Spaces	421
<i>Hernán Aguirre and Kiyoshi Tanaka</i>	
An Evolutionary Method for Natural Language to SQL Translation	432
<i>Alexandre Afonso, Leonardo Brito, and Oto Vale</i>	
Attributes of Dynamic Combinatorial Optimisation	442
<i>Philipp Rohlfshagen and Xin Yao</i>	
A Weighted Local Sharing Technique for Multimodal Optimisation	452
<i>Grant Dick and Peter A. Whigham</i>	

Hybrid Learning

Hybrid Genetic Programming for Optimal Approximation of High Order and Sparse Linear Systems	462
<i>Jing Liu, Wenlong Fu, and Weicai Zhong</i>	
Genetic Vector Quantizer Design on Reconfigurable Hardware	473
<i>Ting-Kuan Lin, Hui-Ya Li, Wen-Jyi Hwang, Chien-Min Ou, and Sheng-Kai Weng</i>	

Pattern Learning and Decision Making in a Photovoltaic System	483
<i>Rongxin Li and Peter Wang</i>	
Using Numerical Simplification to Control Bloat in Genetic Programming	493
<i>David Kinzett, Mengjie Zhang, and Mark Johnston</i>	
Horn Query Learning with Multiple Refinement	503
<i>Josefina Sierra and Josefina Santibáñez</i>	
Evolving Digital Circuits in an Industry Standard Hardware Description Language	514
<i>Jamie Cullen</i>	
Parameterised Indexed FOR-Loops in Genetic Programming and Regular Binary Pattern Strings	524
<i>Gayan Wijesinghe and Vic Ciesielski</i>	
Hierarchical Fuzzy Control for the Inverted Pendulum over the Set of Initial Conditions	534
<i>Juliusz Zajackowski and Brijesh Verma</i>	
Genetic Programming for Feature Ranking in Classification Problems . . .	544
<i>Kourosh Neshatian, Mengjie Zhang, and Peter Andreae</i>	
Time Series Prediction with Evolved, Composite Echo State Networks	555
<i>Russell Y. Webb</i>	
Adaptive Systems	
Genetic Synthesis of Software Architecture	565
<i>Outi Riih�, Kai Koskimies, and Erkki M�kinen</i>	
Dual Phase Evolution and Self-organisation in Networks	575
<i>Greg Paperin, David G. Green, and Tania G. Leishman</i>	
Heterogeneous Payoffs and Social Diversity in the Spatial Prisoner’s Dilemma Game	585
<i>Golriz Rezaei and Michael Kirley</i>	
Theoretical Issues in Evolutionary Computation	
Crossover Can Be Constructive When Computing Unique Input Output Sequences	595
<i>Per Kristian Lehre and Xin Yao</i>	

Real-World Applications of Evolutionary Computation Techniques

Power Electronic Circuits Design: A Particle Swarm Optimization Approach	605
<i>Jun Zhang, Yuan Shi, and Zhi-Hui Zhan</i>	
Computational Intelligence in Radio Astronomy: Using Computational Intelligence Techniques to Tune Geodesy Models	615
<i>Daniel Angus and Adam Deller</i>	
An Efficient Hybrid Algorithm for Optimization of Discrete Structures	625
<i>Amitay Isaacs, Tapabrata Ray, and Warren Smith</i>	
Evolutionary Multi-Objective Optimization for Biped Walking	635
<i>Toshihiko Yanase and Hitoshi Iba</i>	
A Method for Assigning Men and Women with Good Affinity to Matchmaking Parties through Interactive Evolutionary Computation . . .	645
<i>Sho Kuroiwa, Yoshihiro Murata, Tomoya Kitani, Keiichi Yasumoto, and Minoru Ito</i>	
Author Index	657

Modelling Behaviour Cycles for Life-Long Learning in Motivated Agents

Kathryn Merrick

School of Information Technology and Electrical Engineering
University of New South Wales, Australian Defence Force Academy
Northcott Drive, Canberra 2600, Australia
k.merrick@adfa.edu.au

Abstract. Natural systems such as plants, animals and humans exhibit behaviour that forms distinct, rhythmic cycles. These cycles permit individuals and societies to learn, adapt and evolve in complex, dynamic environments. This paper introduces a model of behaviour cycles for artificial systems. This model provides a new way to conceptualise and evaluate life-long learning in artificial agents. The model is demonstrated for evaluating the sensitivity of motivated reinforcement learning agents. Results show that motivated reinforcement learning agents can learn behaviour cycles that are relatively robust to changes in motivation parameters.

Keywords: Behaviour cycles, motivation, life-long learning, reinforcement learning, sensitivity.

1 Introduction

Complex behavioural rhythmicity is thought to have emerged as a result of generations of evolution in individuals and societies [1]. Plants, animals and humans display behaviour cycles that vary in duration from minutes to days and months. These cycles occur at biological, cognitive and social levels to connect an individual to its environment, drive exploration and force behavioural evolution. This paper presents a model of behaviour cycles for artificial systems. By formalising the concept of a behaviour cycle, the model provides a new way to conceptualise and evaluate life-long learning in artificial agents, with a view to the design of agents that can exhibit life-long adaptability, exploration and social evolution through self-motivation and learning.

The remainder of this section reviews the evolution of behaviour cycles in natural systems and describes the potential advantages of behaviour cycles in artificial systems. Section 2 introduces a model of behaviour cycles for artificial systems. Section 3 uses the new model of behaviour cycles to evaluate behaviour cycles in motivated reinforcement learning agents. Results show that motivated reinforcement learning agents exhibit behaviour cycles that are relatively robust to changes in motivation parameters.

1.1 Biological Cycles

Biological cycles, biorhythms or biocycles are closely associated with both the physical form of a natural organism and its environment [1]. Environmental factors influencing the evolution of biocycles include seasonal cycles in temperature and light [5], variations in the earth's magnetic field [22] and tidal cycles [3]. Biocycles may also be influenced internally by hormonal or chemical cycles.

Ahlegren and Halberg [1] propose four advantages of biological behaviour cycles in natural systems: anticipation of environmental change, efficiency of action, competition reduction through exploitation of environmental niches and navigation. Designing artificial systems that are capable of evolving emergent behaviour cycles has the potential to draw on these advantages. In particular, it provides a basis for building systems that can synchronise their behaviour with their environment to achieve, efficient, long term activity through anticipation and competitive behaviour.

1.2 Cognitive Cycles

Where biological cycles are associated with the physical form or environment of an individual, cognitive cycles are associated with abstract reasoning processes. Cycles in risk taking behaviour [4, 24], habituation and recovery [7] and learning [8] are examples of cognitive cycles. Habituation describes the process of stimuli losing novelty for an individual. Loss of novelty motivates exploration and experimentation to achieve a more optimal level of stimulation. In this way, individuals engage in an ongoing cycle of habituation and recovery.

Closely related to habituation and recovery is the learning cycle. Kolb et al. [8] describe a four step learning cycle that moves through phases of concrete experience, reflection and observation, abstract conceptualisation and active experimentation. Depending on the results of experimentation, the cycle starts over with a new learning experience or with a revision of the current one.

Where biological behaviour cycles tend to be closely associated with efficiency and survival, cognitive behaviour cycles are frequently associated with risk taking behaviour, experimentation or creativity. These are all potential advantages of designing artificial systems capable of emergent cognitive behaviour cycles.

1.3 Social Cycles

Cyclic behaviour can be observed in groups of individuals, at both biological and cognitive levels. At the biological level, co-evolution [6] of parasites and their hosts, predators and prey or birds and flowers is an example of a social cycle.

At the cognitive level, Social Cycle Theory describes the evolution of society and human history as progressing through a series of cycles. Various mathematical models of these cycles have been developed [15, 21]. These models demonstrate that socio-demographic cycles are a basic feature of complex agrarian systems for land ownership and labour organisation.

Social cycles can be thought of as describing the rise and fall of species, trends, cultures and societies. These cycles force social progress through a cyclic process of social evolution. Just as the implementation of pseudo biological and cognitive cycles in artificial systems promises advantages for efficiency, survival and creativity, artificial systems that can self-organise social cycles have the advantage of an implicit means of continued self-advancement.

1.4 Behaviour Cycles in Artificial Systems

The idea of developing artificial agents that exhibit behaviour cycles has been considered in a number of domains. For example, the simplest approaches to artificially intelligent characters in computer games use scripted animations that repeat a small set of behaviours as a short cycle [9]. More complex story-telling approaches create longer cycles using role based models [11]. However, both these approaches assume a fixed set of behaviours through which the agent cycles. The agent cannot adapt its behaviour cycles to changes in its environment, beyond those for which it has been programmed.

A number of more adaptive reasoning processes have been developed to achieve life-long learning in artificial agents. Life-long learning and developmental robotics researchers take the view that agents should be equipped for long-term, general purpose learning of any tasks that may arise, and not limited to learning about a limited set of pre-defined tasks [19]. A range of different approaches have been proposed to life-long learning including knowledge transfer [20], intrinsically motivated agents [17, 18], active learning [2] and reinforcement learning [23]. While the idea of behaviour cycles is relevant to life-long learning, the concept has not been formalised in this field.

The idea of behaviour cycles has, however, been considered with reference to motivated reinforcement learning agents controlling non-player characters in games [12-14], and curious agents as immobile robots in built environments [10]. In both cases, motivated reinforcement learning agents provide an alternative to the fixed behaviour cycles of scripted approaches. Existing techniques for modelling behaviour cycles include state machine representations [14] and charts visualising the actions performed by the agent against time [13]. These approaches provide qualitative measurements of behaviour cycles, but are not useful for making quantitative comparisons between agents.

One quantitative approach to measuring behaviour cycles has been to study the standard deviation of the number of actions in a cycle [12]. This can then be used to measure properties such as the number of cycles and the length of cycles. This approach has proved effective for identifying and measuring behaviour cycles in agents situated in simple environments, but it does not take into account the impact of noise in complex environments where agents may develop longer behaviour cycles. This paper presents an alternative model to address this issue. In addition, behaviour cycles are modelled as repeated states and transitions, as well as repeated actions.

2 Modelling Behaviour Cycles

The model in this paper borrows notation from machine learning, which considers an artificial agent in terms of its input and output [16]. The environment of an agent is represented as a set $\mathbf{S}$ of states $S_1, S_2, S_3, \dots$. At any time t , the agent senses (inputs) a state $S_{(t)}$. The agent can perform (output) an action $A_{(t)}$. This causes the environment to make a transition $T_{(t)}$ to a new state, as shown in Figure 1.

A state S may be represented in a variety of ways, including attribute based representations, relational representations or a simple, unique identifier. Transitions represent the difference between two states and their representation depends on the state representation used. Over time, the agent will experience a trajectory of states, actions and transitions as follows:

$$S_1, A_1, T_1, S_2, A_2, T_2, S_3, \dots$$

Thus, different types of behaviour cycles are possible, including cycles of repeated states, actions or transitions in the experience trajectory. These are discussed below.

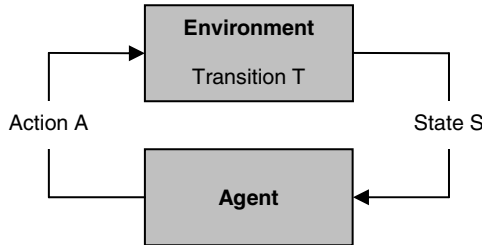


Fig. 1. At each time t , an agent senses a state of the environment S and chooses an action A . This causes the environment to undergo transition T .

2.1 Types of Behaviour Cycles

Cycles of Repeated States: A cycle of repeated states is a sequence of states in which one or more states is repeated, as shown in Figure 2(a). A cycle of repeated states may or may not include repeated actions and transitions. Cycles of repeated states might emerge in migratory agents for example, which travel to different locations at different times.

Cycles of Repeated Transitions: A cycle of repeated transitions, shown in Figure 2(b), may occur in conjunction with a cycle or repeated states, or without a cycle of repeated states. For example an agent making something will cause the environment to transition in the same way for each intermediate step, but when the new object is created there will be progressively more of that object, resulting in unique states in each cycle.

Cycles of Repeated Actions: An agent may also develop a cycle of repeated actions as shown in Figure 2(c). Measuring cycles of repeated actions is most likely to be meaningful in predictable environments where the outcome of each action is guaranteed. In unpredictable environments, actions may be ineffective so cyclic action sequences have no impact on the agent or environment.

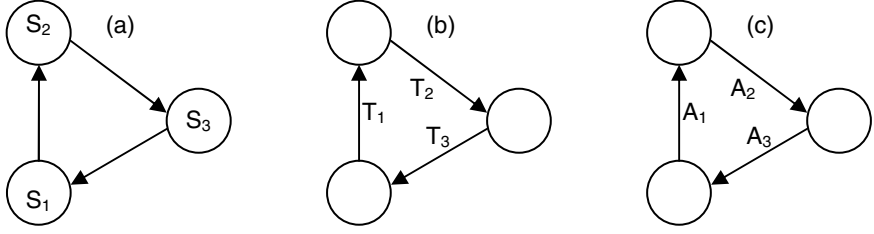


Fig. 2. Different types of behaviour cycles for artificial agents: (a) Cycle of repeated states, (b) cycle of repeated transitions and (c) cycle of repeated actions

2.2 A Statistical Model for Behaviour Cycles

For the purpose of measuring behaviour cycles, the repeated state, action or transition is denoted K . A behaviour cycle for K may occur just once or it may occur many times. A cycle that occurs just once may be a result of random chance, while a behaviour cycle that is repeated many times is more likely to represent a stable, emergent behaviour cycle. Previously, standard deviation has been used to identify stable behaviour cycles [12]. However, using this measure unfairly penalises long behaviour cycles in noisy environments. To overcome this issue, a dimensionless value is required to permit meaningful conclusions to be drawn about the emergence of stable behaviour cycles. One method for achieving this is to use the coefficient of variation to measure the number of states, actions or transitions in a behaviour cycle.

To analyse behaviour cycles for a state, action or transition K , the last h repetitions of K are considered. The coefficient of variation for the number of actions between repetitions of K is then computed:

$$c_v(K) = \frac{\sqrt{\frac{1}{h-1} \sum_{i=1}^h (a_i - \bar{a}_K)^2}}{\bar{a}_K} \quad (1)$$

$\bar{a}_K$ is the mean number of actions required to repeat K during the last h successive repetitions of K . The coefficient of variation represents a normalised measure of the dispersion of the number of actions about the mean $\bar{a}_K$. A behaviour cycle for K is stable when it has been repeated h times with $c_v(K)$ less than some error threshold r . $h=20$ and $r=0.2$ are used in this paper, which allows for approximately 20% noise in a behaviour cycle repeated 20 times.

Merrick [12] introduced two metrics, behavioural variety and behavioural complexity, for characterising behaviour cycles. The following sections provide adaptations of these metrics, using the coefficient of variation to identify behaviour cycles.

Behavioural Complexity: Natural systems may exhibit behaviour cycles varying in length from minutes to days or months. The behavioural complexity metric characterises artificial agents in terms of the length of their behaviour cycles. The length or complexity C of a behaviour cycle for K can be measured as the mean numbers of actions $\bar{a}_K$ required to repeat K :

$$C_K = \bar{a}_K \quad (2)$$

Agents may be characterised in terms of their minimum, average or maximum behavioural complexity. A high maximum behavioural complexity is particularly important for agents to be able to function in complex environments.

Behavioural Variety: An agent may develop behaviour cycles for many different states, transitions or actions. Behavioural variety evaluates the behaviour of an agent by counting the number of different states, transitions or actions for which behaviour cycles emerge. This measurement is made by analysing the agent's experience trajectory at time t , and counting distinct states, transitions or actions K for which $c_v(K) < r$, as shown in Equation 3. The ability to develop high behavioural variety is important for agents that are to function in dynamic environments

$$V_{(t)} = \text{count\_unique} (K) \text{ where } c_v(K) < r \quad (3)$$

3 Behaviour Cycles in Motivated Agents

Motivated reinforcement learning (MRL) [12-14] is a promising approach to the design of agents that can evolve behaviour cycles. This is because the motivation component autonomously adapts the focus of attention in response to the agent's experiences in its environment. Behaviour cycles emerge as the reinforcement learning component uses trial-and-error to learn about the current focus of attention. However, one criticism of MRL is that it is difficult to select values for the parameters of the motivation function, and it is unclear how parameter selection affects performance. This section uses the new model of behaviour cycles from Section 2.2 to analyse the sensitivity of an existing MRL model to changes in its motivation parameters. Results indicate that MRL is a promising starting point for developing artificial agents behaviour cycles that can adapt to multiple tasks.

3.1 Experimental Setup

Merrick and Maher [12-14] model MRL using a computational model of interest as motivation. Interest is modelled as the sum of two feedback functions for positive and negative responses to the novelty of a transition T . The equations for interest I and the novelty N_T of a transition T are shown in Equations 4 and 5 respectively.

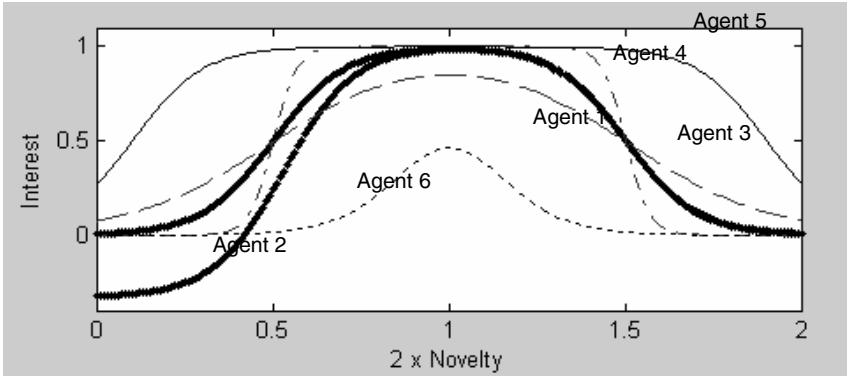
The shape of the interest curve depends on six parameters of the feedback functions, as well as parameters of the novelty curve. Merrick and Maher [12-14] choose values for these parameters arbitrarily to form a smooth interest curve. However, clearly other values are also possible. Table 1 lists five alternative sets of interest parameter values that are used to test the sensitivity of MRL in this paper. The resulting interest curves are shown in Figure 3. For simplicity, in this paper, the novelty parameters are fixed at $\tau=14.3$, $\alpha=1.05$ and $\zeta_{(t)} = 1$.

$$I = \frac{F_{\max}^+}{C_1 + e^{-\rho^+ (2N_{T(t)} - F_{\min}^+)}} - \frac{F_{\max}^-}{C_2 + e^{-\rho^- (2N_{T(t)} - F_{\min}^-)}} - C_3 \text{ where} \quad (4)$$

$$\tau \frac{dN_{T(t)}}{dt} = \alpha [N_{T(0)} - N_{T(t)}] - \zeta_{(t)} \quad (5)$$

Table 1. Alternative parameter values for experimental agents

Agent	Parameters	Description
1	$F_{\max}^+ = F_{\max}^- = 1, \rho^+ = \rho^- = 10, F_{\min}^+ = 0.5, F_{\min}^- = 1.5, C_1 = C_2 = 1, C_3 = 0$	Merrick and Maher [12-14]
2	$F_{\max}^+ = F_{\max}^- = 1, \rho^+ = \rho^- = 10, F_{\min}^+ = 0.5, F_{\min}^- = 1.5, C_1 = 0.75, C_2 = 1, C_3 = 0.33$	Aversion to low novelty
3	$F_{\max}^+ = F_{\max}^- = 1, \rho^+ = \rho^- = 5, F_{\min}^+ = 0.5, F_{\min}^- = 1.5, C_1 = C_2 = 1, C_3 = 0$	Slow learning
4	$F_{\max}^+ = F_{\max}^- = 1, \rho^+ = \rho^- = 30, F_{\min}^+ = 0.5, F_{\min}^- = 1.5, C_1 = C_2 = 1, C_3 = 0$	Fast Learning
5	$F_{\max}^+ = F_{\max}^- = 1, \rho^+ = \rho^- = 10, F_{\min}^+ = 0.1, F_{\min}^- = 1.9, C_1 = C_2 = 1, C_3 = 0$	Long attention span
6	$F_{\max}^+ = F_{\max}^- = 1, \rho^+ = \rho^- = 10, F_{\min}^+ = 0.9, F_{\min}^- = 1.1, C_1 = C_2 = 1, C_3 = 0$	Short attention span

**Fig. 3.** Variations in the interest curve used to analyse the sensitivity of motivated reinforcement learning to changes in the values of motivation parameters

Merrick [12] uses a simple game scenario to analyse MRL. In the scenario, non-player characters have access to tools they can use to acquire raw materials and convert these materials to finished products. The tools are a pick, an axe, a smelt, a forge and a lathe. Raw materials are wood, iron ore and iron. Finished products are furniture and weapons. This scenario requires one unit of raw material to make a finished product. In contrast, the experiments in this paper use a more complex version of the environment, in which five units of a raw material are required to make a finished product.

Each type of agent was run 20 times for 50,000 time-steps in this environment. As the interest function computes novelty in terms of transitions, results in this section are computed using $K=T$ for all T . While life-long learning generally implies a longer lifetime, fixing the lifetime in this case permits a fair analysis of the agents' behaviour cycles. Results show the 95% confidence interval.

3.2 Sensitivity of Motivated Reinforcement Learning

Figure 4(a) show that the six MRL agents develop behaviour cycles repeating between 11 and 17 transitions. This means that each agent learns behaviour cycles for 11-17 different tasks. Two examples of learned behaviour cycles are shown in Figure 5. In this environment, behaviour cycles are quite robust in terms of behavioural variety, with only Agent 2 showing a statistically significant decrease in behavioural variety when compared to the other agents. However, Agent 2, Agent 4 and Agent 6 evolve the most complex behaviour cycles, as shown in Figure 4(b). Agent 3 and Agent 5 show significantly lower behavioural complexity, most likely due to lower expressiveness of these curves. (The curve for Agent 3 changes very slowly and the curve for Agent 5 has a reduced range.)

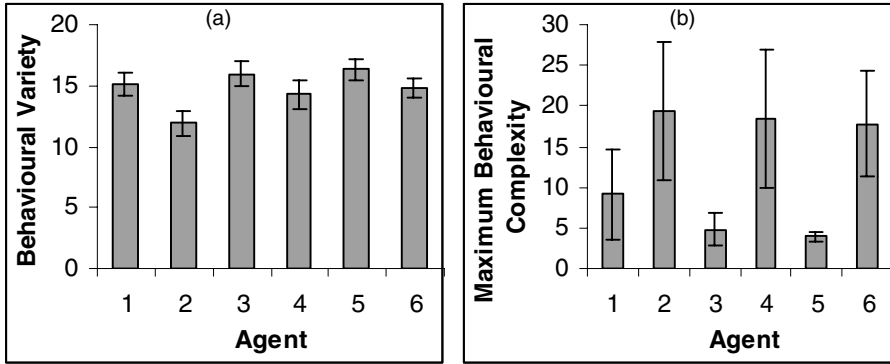


Fig. 4. Behaviour cycle characteristics of six agents compared using (a) behavioural variety and (b) behavioural complexity

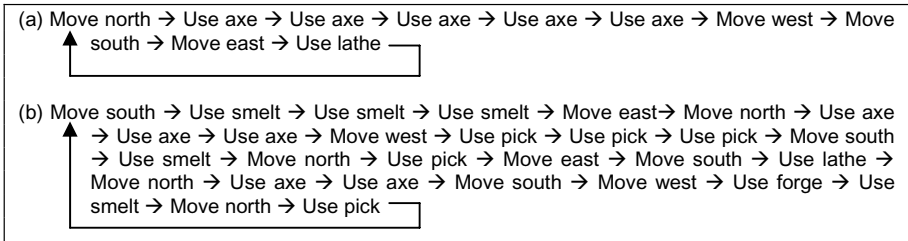


Fig. 5. Examples of behaviour cycle evolved by: (a) Agent 6: 11 actions to collect wood and make furniture. (b) Agent 2: 29 actions to move, gather raw materials and use the lathe, smelt and forge to make furniture and weapons.

Figure 5 describes two sample behaviour cycles learned by different agents. It shows how different agents evolve different behaviour cycles. This is useful in multi-agent simulations as individual agents do not have to be pre-programmed. Rather different behaviour cycles are evolved in response to an agent's experiences in its environment. Figure 5(b) is interesting because it shows how the agent can interleave steps for different activities into a long behaviour cycle. This represents a game character that learns to be a weapon-smith and carpenter. Such behaviour may be relevant to 'trainer' characters in games such as *World of Warcraft*, which currently tend to stand in one place rather than engaging in the activity they purport to be experts in.

4 Conclusion and Future Work

This paper has presented a model of behaviour cycles for artificial systems. This model provides a new way to conceptualise and evaluate life-long learning in artificial agents. The model is demonstrated for evaluating the sensitivity of motivated reinforcement learning agents to changes in their motivation parameters. Results show that motivated reinforcement learning agents exhibit behaviour cycles that are quite robust to changes in motivation parameters. Behavioural variety does not vary significantly between agents. However, agents with an aversion to low novelty can develop more complex behaviour cycles than agents using other forms of the interest curve.

The results in this paper provide a simple demonstration of the concept of behaviour cycles in artificial systems. However, future work is required in various areas. In particular, it is unclear from the results so far whether motivated learning agents are capable of displaying emergent behaviour cycles more complex and realistic virtual world or real world scenarios. Further experiments are required to analyse this. The development of new motivation modules may be required to achieve greater behavioural variety or complexity in such environments.

In the long term, the concept of behaviour cycles is likely to be relevant to research in fields such as life-long learning, developmental robotics and computer games, which require adaptive agents that can solve a range of problems that arise over extended time periods. In future, new agent models are envisaged that permit the evolution of complex behaviour cycles in complex, dynamic environments. Thus, further advantage is to be gained by specifically modelling behaviour cycles, analysing the potential of existing agent models for achieving emergent behaviour cycles cyclic and developing new agent models that explicitly exploit cyclic behaviour at biological, cognitive and social levels.

References

1. Ahlgren, A., Halberg, F.: Cycles of nature: an introduction to biological rhythms. National Teachers Association, Washington (1990)
2. Bondu, A., Lemaire, V.: Active learning using adaptive curiosity. In: Proceedings of the Seventh International Conference on Epigenetic Robotics, Lund University (2007)
3. Brown, F.A.: Persistent activity rhythms in the oyster. *American Journal of Physiology* 178(33), 510–514 (1954)

4. Chavanne, T.: Variation in risk taking behaviour among female college students as a function of the menstrual cycle. *Evolution and Human Behaviour* 19(1), 27–32 (2003)
5. Dunlap, J., Loros, J., De Coursey, P.: *Chronobiology: biological timekeeping*. Sinauer Associates (2003)
6. Ehrlich, P., Raven, P.: Butterflies and plants: a study in co-evolution. *Evolution* 18, 584–608 (1964)
7. Green, R.G., Beatty, W.W., Arkin, R.M.: *Human motivation: physiological, behavioural and social approaches*. Allyn and Bacon, Inc., Massachussets (1984)
8. Kolb, D.A., Rubin, I.M., McIntyre, J.M. (eds.): *Organizational Psychology: Readings on Human Behaviour in Organizations*. Prentice-Hall, Englewood Cliffs (1984)
9. Laird, J., van Lent, M.: Interactive computer games: human-level AI's killer application. In: *National Conference on Artificial Intelligence (AAAI)*, pp. 1171–1178 (2000)
10. Macindoe, O., Maher, M.L., Merrick, K.: Agent based intrinsically motivated intelligent environments, *Handbook on Mobile and Ubiquitous Computing: Innovations and Perspectives*. American Scientific Publishers (2008)
11. Mac Namee, B., Dobbyn, S., Cunningham, P., O'Sullivan, C.: Simulating virtual humans across diverse situations. In: Rist, T., Aylett, R.S., Ballin, D., Rickel, J. (eds.) *IVA 2003*. LNCS, vol. 2792, pp. 159–163. Springer, Heidelberg (2003)
12. Merrick, K.: Modelling motivation for adaptive non-player characters in dynamic computer game worlds. *ACM Computers in Entertainment* 5(4) (2007)
13. Merrick, K., Maher, M.L.: Motivated reinforcement learning for non-player characters in persistent computer game worlds. In: *ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, Los Angeles, USA (2006)
14. Merrick, K., Maher, M.L.: Motivated reinforcement learning for adaptive characters in open-ended simulation games. In: *ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, Salzburg, Austria, pp. 127–134 (2007)
15. Nefedov, S.A.: A Model of Demographic Cycles in Traditional Societies: The Case of Ancient China. *Evolution and History* 3(1), 69–80 (2004)
16. Nilsson, N.: Introduction to machine learning (accessed January 2006), <http://ai.stanford.edu/people/nilsson/mlbook.html>
17. Oudeyer, P.Y., Kaplan, F., Hafner, V.: Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation* 11(2), 265–286 (2007)
18. Singh, S., Barto, A.G., Chentanez, N.: Intrinsically motivated reinforcement learning. In: *Advances in Neural Information Processing Systems 17 (NIPS)*, pp. 1281–1288 (2005)
19. Stoytchev, S.: Five basic principles of developmental robotics. In: *NIPS Workshop on Grounding, Perception, Knowledge, and Cognition* (2006)
20. Thrun, S., Mitchell, T.: *Lifelong robot learning*. Robotics and Autonomous Systems (1993)
21. Usher, D.: The dynastic cycle and the stationary state. *The American Economic Review* 79, 1031–1044 (1989)
22. Wever, R.: Human circadian rhythms under the influence of weak electric fields and the different aspects of these studies. *Intn'l Journal of Biometeorology* 17(3), 227–232 (1973)
23. Winberg, S., Balkenius, C.: Generalization and specialization in reinforcement learning. In: *Proceedings of the 7th Int'l Conference on Epigenetic Robotics*, Lund University (2007)
24. Zimecki, M.: The lunar cycle: effects on human and animal behaviour and physiology. *Postepy Hig. Med. Dosw.* 60, 1–7 (2006)

Breaking the Synaptic Dogma: Evolving a Neuro-inspired Developmental Network

Gul Muhammad Khan, Julian F. Miller, and David M. Halliday

Electronics Department, University of York, York, YO10 5DD, UK

`drgul@cgpnuron.com`, `{jfm,dh20}@ohm.york.ac.uk`

<http://www.york.ac.uk>

Abstract. The majority of artificial neural networks are static and lifeless and do not change themselves within a learning environment. In these models learning is seen as the process of obtaining the strengths of connections between neurons (i.e. weights). We refer to this as the 'synaptic dogma'. This is in marked contrast with biological networks which have time dependent morphology and in which practically all neural aspects can change or be shaped by mutual interactions and interactions with an external environment. Inspired by this and many aspects of neuroscience, we have designed a new kind of neural network. In this model, neurons are represented by seven evolved programs that model particular components and aspects of biological neurons (dendrites, soma, axons, synapses, electrical and developmental behaviour). Each network begins as a small randomly generated network of neurons. When the seven programs are run, the neurons, dendrites, axons and synapses can increase or decrease in number and change in interaction with an external environment. Our aim is to show that it is possible to evolve programs that allow a network to learn through experience (i.e. encode the *ability* to learn). We report on our continuing investigations in the context of learning how to play checkers.

1 Introduction

Artificial Neural Networks (ANNs), though inspired by the brain have largely ignored many aspects of biological neural systems. Originally, there were good reasons for this. Simple models were required that could be executed on relatively slow computers. However, the computational power of modern computers has made more complex neuro-inspired approaches much more feasible. At the same time, our understanding of neuroscience has increased considerably. In our view two of the most important aspects that we should consider incorporating are neural development and time-dependent morphology. Marcus argues convincingly about the importance of development “mechanisms that build brains are just extensions of those that build the body” [1]. It is also becoming more apparent that sub-processes of neurons are highly time-dependent so that many structures are in a constant state of being re-built and changed [2]. Indeed memory itself is not a static process and the location and mechanisms responsible for

remembered information is in constant (though, largely gradual) change. The act of remembering is a process of reconstructing and changing the original structure that was associated with the original event [3].

Various studies have shown that “Dendritic trees enhance computational power” [4]. Neurons communicate through synapses which are not merely the point of connection between neurons [5]. They can change the strength and shape of the signal over various time scales. We have taken the view that the time dependent and environmentally sensitive variation of morphology and many other processes of real neurons is very important and models are required that incorporate these features. In our model a neuron consists of a soma, dendrites, axons with branches and dynamic synapses and synaptic communication. Neurite branches can grow, shrink, self-prune, or produce new branches to arrive at a network whose structure and complexity is related to properties of the learning problem [6].

Our aim is to find a set of computational functions that can allow the above characteristics and in so doing, allow us to build a neural network that is capable of learning through experience. Thus we aim to find a way of giving the network an *ability* to learn, so that by repeated experience a network can improve its capability to solve a range of problems. Such a network would be very different from conventional ANN models as the network would be self-training and constantly adjust itself over time in response to external environmental signals.

From our studies of neuroscience, we have identified seven essential computational functions that need to be included in a model of a neuron and its communication mechanisms [7]. From this we decided what kind of data these functions should work with and how they should interact, however we cannot design the functions themselves. So we turned to an automatic method of program design, namely, Genetic Programming [8] to help us with this problem. In particular, we have used a well established and efficient form of Genetic Programming called Cartesian Genetic Programming (CGP) in which programs are represented by directed acyclic graphs [9]. In CGP the genotype is a fixed length list of integers, which encode the function of nodes and the connections of a directed graph.

In order to evaluate the effectiveness of this approach we have previously applied it to a classic AI problem called wumpus world [10]. We found that the agents improved with experience and exhibited a range of intelligent behaviours. In this paper we have turned our attention to a much more serious problem in AI that has been studied ever since the beginnings of AI, namely computer checkers. The approach taken in AI to obtaining programs that could play such games at a high level was originally developed by Shannon [11]. He developed the idea of using a game tree of a certain depth and advocated using a *board evaluation function*. This function allocates a numerical score according to how good a board position is for a player. Once this is obtained a method of minimax can be used [12]. This works by propagating scores up the game tree to determine the optimum move for either player.

Since we are interested in how a single computational agent can learn how to play checkers *in its lifetime* we have not used a board evaluation function

or minimax. Instead our checkers playing agents start from a small random network at the beginning of each game and through the running of its neural developmental programs builds a network that makes its moves. In other words, our agents grow up playing checkers

The plan of the paper is as follows. Section 2 gives a discussion of some important aspects of neuroscience that we have included in our computational network. Section 3 describes the CGP computational network (CGPCNN). In section 4 we present our results and findings and in section 5 our concluding remarks and observations.

2 The CGP Computational Network (CGPCN)

This section describes in detail the structure of the CGPCN, along with the rules and evolutionary strategy used to run the system.

In the CGPCN neurons are placed randomly in a two dimensional spatial grid so that they are only aware of their spatial neighbours (as shown in figure 1). Each neuron is initially allocated a random number of dendrites, dendrite branches, one axon and a random number of axon branches. Neurons receive information through dendrite branches, and transfer information through axon branches to neighbouring neurons.

The dynamics of the network also change since branches may grow or shrink and move from one CGPCN grid point to another. They can produce new branches and can disappear, and neurons may die or produce new neurons. Axon branches transfer information only to dendrite branches in their proximity. Electrical potential is used for internal processing of neurons and communication between neurons and we represent it as an integer.

Health, Resistance, Weight and Statefactor

Four variables are incorporated into the CGPCN, representing either fundamental properties of the neurons (*health*, *resistance*, *weight*) or as an aid to computational efficiency (*statefactor*). The values of these variables are adjusted by the CGP programs. The *health* variable is used to govern replication and/or death of dendrites and connections. The *resistance* variable controls growth and/or shrinkage of dendrites and axons. The *weight* is used in calculating the potentials in the network. Each soma has only two variables: *health* and *weight*.

The *statefactor* is used as a parameter to reduce computational burden, by keeping some of the neurons and branches inactive for a number of cycles. Only when the *statefactor* is zero the neurons and branches are considered to be active and their corresponding program is run. The value of the *statefactor* is affected indirectly by CGP programs.

Inputs, Outputs and Information Processing in the Network

The external inputs (encoding a simulated potential) are applied to the CGPCN and processed by the axo-synapse program, AS. These are distributed in the network in a similar way to the axon branches of neurons. When AS is executed,

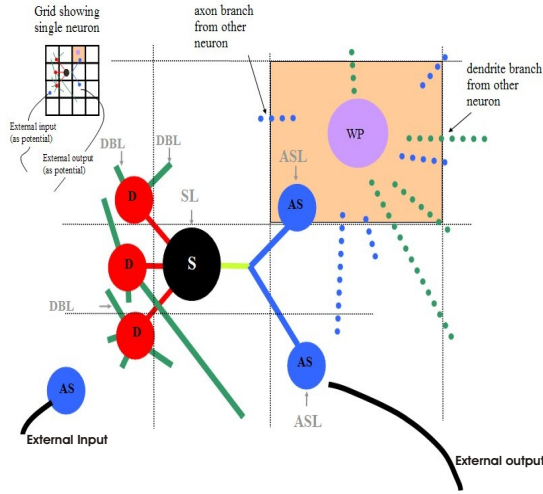


Fig. 1. On the top left a grid is shown containing a single neuron. The rest of the figure is an exploded view of the neuron is given. The neuron consists of seven evolved computational functions. Three are electrical and process a simulated potential in the dendrite (D), soma (S) and axo-synapse branch (AS). Three more are developmental in nature and are responsible for the life-cycle of neural components (shown in grey). They decide whether dendrite branches (DBL), soma (SL) and axo-synaptic branches (ASL) should die, change, or replicate. The remaining evolved computational function (WP) adjusts synaptic and dendritic weights and is used to decide the transfer of potential from a firing neuron (dashed line emanating from soma) to a neighbouring neuron.

it modifies the potentials of neighbouring active dendrite branches. We obtain output from the CGPCN via dendrite branches. These branches are updated by the AS program of neurons and after five cycles the potentials produced are averaged and this value (Fig 1) is used as the external output.

Information processing in the network starts by selecting the list of active neurons in the network and processing them in a random sequence. Each neuron take the signal from the dendrites by running the electrical dendrite branch program. The signals from dendrites are averaged and applied to the soma program along with the soma potential. The soma program is executed to get the final value of soma potential, which decides whether a neuron should fire an action potential or not. If soma fires, an action potential is transferred to other neurons through axosynaptic branches. The same process is repeated in all neurons.

CGP Model of Neuron

In our model neural functionality is divided into three major categories: electrical processing, life cycle and weight processing. These categories are described in detail below.

Electrical Processing

The electrical processing part is responsible for signal processing inside neurons and communication between neurons. It consists of dendrite branch, soma, and axo-synaptic branch electrical chromosomes.

The dendrite program D handles the interaction of dendrite branches belonging to a dendrite. It takes active dendrite branch potentials and soma potential as input and updates their values. The *Statefactor* is decreased if the update in potential is large and vice versa. If any of the branches are active (has its statefactor equal to zero), their life cycle program (DBL) is run, otherwise it continues processing the other dendrites.

The soma program S determines the final value of soma potential after receiving signals from all the dendrites. The processed potential of the soma is then compared with the threshold potential of the soma, and a decision is made whether to fire an action potential or not. If it fires, it is kept inactive (refractory period) for a few cycles by changing its *statefactor*, the soma life cycle chromosome (SL) is run, and the firing potential is sent to the other neurons by running the AS programs in axon branches.

AS updates neighbouring dendrite branch potentials and the axo-synaptic potential. The *statefactor* of the axosynaptic branch is also updated. If the axo-synaptic branch is active its life cycle program (ASL) is executed.

After this the weight processing program (WP) is run which updates the *Weights* of neighbouring (branches sharing same grid square) branches. The processed axo-synaptic potential is assigned to the dendrite branch having the *largest* updated *Weight*.

Life Cycle of Neuron

This part is responsible for replication or death of neurons and neurite branches and also the growth and migration of neurite branches. It consists of three life cycle chromosomes responsible for the neuron and neurites development.

The two branch chromosomes update *Resistance* and *Health* of the branch. Change in *Resistance* of a neurite branch is used to decide whether it will grow, shrink, or stay at its current location. The updated value of neurite branch *Health* decides whether to produce offspring, to die, or remain as it was with an updated *Health* value. If the updated *Health* is above a certain threshold it is allowed to produce offspring and if below certain threshold, it is removed from the neurite. Producing offspring results in a new branch at the same CGPCN grid point connected to the same neurite (axon or dendrite).

The soma life cycle chromosome produces updated values of *Health* and *Weight* of the soma as output. The updated value of the soma *Health* decides whether the soma should produce offspring, should die or continue as it is. If the updated *Health* is above certain threshold it is allowed to produce offspring and if below a certain threshold it is removed from the network along with its neurites. If it produces offspring, then a new neuron is introduced into the network with a random number of neurites at a different random location.

3 Experimental Setup

The experiment is organized such that an agent is provided with CGPCN as its computational network and is allowed to play five games against a minimax based checker program (MCP). The initial population of five agents (Evolutionary strategy $1+\lambda$, with λ set to 4) each starting with a small randomly generated initial network and randomly generated genotypes. The agents each play five games of checkers against the MCP. So in every game the agent starts with a developed network from previous (unless it is the first game in which case it begin with a random network) game and is allowed to continue to develop during the five game series. The genotype corresponding to the agent with the highest average fitness at the end of five games is selected as the parent for the new population. Four offspring formed by mutating the parent are created. Any learning behaviour that is acquired by an agent is obtained through the interaction and repeated running of the seven chromosomes within the game scenario.

The MCP always plays the first move. The updated board is then applied to an agent's CGPCN. The potentials representing the state of the board are applied to CGPCN using the axosynapse(AS) chromosome. Input is in the form of board values, which is an array of 32 elements, with each representing a playable board square. Each of the 32 inputs represents one of the following five different values depending on what is on the square of the board(represented by I). The values taken by I are as follows: if empty $I=0$, if king $I=\text{Maximum value}(M) 2^{32} - 1$, if piece $I=(3/4)M$, if opposing piece $I=(1/2)M$, and finally if opposing king, $I=(1/4)M$. The board inputs are applied in pairs to all the sixteen locations in the 4x4 CGPCN grid (i.e. two virtual axo-synapse branches in every grid square). The CGPCN is then run for five cycles. During this process it updates the potentials of the dendrite branches acting as output of the network. Output is in two forms, one of the outputs is used to select the piece to move and second is used to decide where that piece should move. Each piece on the board has an output dendrite branch in the CGPCN. All pieces are assigned a unique ID, representing the CGPCN grid square where its branch is located. Each of these branches has a potential, which is updated during CGPCN processing. The values of potentials determine the possibility of a piece to move, the piece that has the highest potential will be the one that is moved, however if any pieces are in a position to jump then the piece with the highest potential of those will move. Note that if the piece is a king and can jump then, according to the rules of checkers, this takes priority. Once again if two pieces are kings and each could jump the king with the highest potential makes the jumping move. In addition, there are also five output dendrite branches distributed at random locations in the CGPCN grid. The average value of these branch potentials determine the direction of movement for the piece. Whenever a piece is removed its dendrite branch is removed from the CGPCN grid.

The game is stopped if either the CGPCN of an agent or its opponent dies (i.e. all its neurons or neurites dies), or if all its or opponent players are taken,

or if the agent or its opponent can not move anymore, or if the allotted number of moves allowed for the game have been taken.

CGP Computational Network (CGPCN) Setup

The CGPCN is arranged in the following manner for this experiment. Each player CGPCN has neurons and branches located in a 4x4 grid. Initial number of neurons is 5. Maximum number of dendrites is 5. Maximum number of dendrite and axon branches is 200. Maximum branch *statefactor* is 7. Maximum soma *statefactor* is 3. Mutation rate is 5%. Maximum number of nodes per chromosome is 200. Maximum number of moves is 20 for each player.

Fitness Calculation

Both the agent and the software is allowed to play a limited number of moves and their fitness is accumulated at the end of this period using the following equation:

$Fitness = A + 200N_K + 100N_M - 200N_{OK} - 100N_{OM} + N_{MOV}$, Where N_K represents the number of kings, and N_M represents number of men of the current player. N_{OK} and N_{OM} represent the number of kings and men of the opposing player. N_{MOV} represents the total number of moves played. A is 1000 for a win, and zero for a draw. To avoid spending much computational time assessing the abilities of poor game playing agents we have chosen a maximum number of moves. If this number of moves is reached before either of the agents win the game, then $A = 0$, and the number of pieces and type of pieces decide the fitness value of the agent.

3.1 Results and Analysis

From the fitness graph shown in figure 2 (left), it is difficult to assess if any learning has taken place, because the MCP is playing at a much higher level and the evolved agent learns from a random network in game 1 which is developed by the evolved programs over the five game series. It seems quite difficult to learn from a highly skilled system in general.

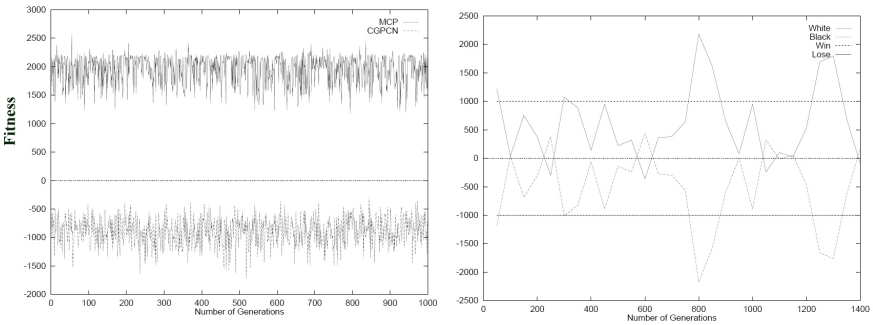


Fig. 2. Fitness of CGPCN playing checkers against MCP over the course of evolution (left), and average fitness variation of high evolved agent against less evolved agent

We tested whether the agent's level of play got better with evolution or not. We tested them against less evolved agents and found that the well evolved agent almost always beats the less evolved one, in some of the cases it ends up in a draw, but in those cases the well evolved agent ends up with more kings and pieces than the less evolved agent.

Figure 2(right) shows the variation in fitness of a well evolved agent (2000th generation) playing white against, less evolved agents from various generations (140 games). From the graph it is clearly visible that the highly evolved agent always beat the less evolved ones. However, the well evolved player appears to play better against players from later generation rather than earlier. The reason for this are as follows: the MCP updates its database after every game and plays a completely different game even against the same opponent. This means that it is difficult for the agent to maintain its previously gained higher value of fitness on a subsequent game. In order to achieve a higher fitness, the agent has to play slightly different game from its earlier games. When an evolved player plays against close ancestral relatives they tend to play in a similar way making it easy for the well evolved agent to beat it as evident from figure 2(right). Whereas they find it difficult to win, when their opponents are distant ancestral relative, since they play in a very different way.

In a further experiment we studied the performance of evolved agents over a 500 games sequence. One agent began each game with a random network and the other (more evolved) was allowed to retain its network and develop it over all the games. A well evolved agent from generation 2000 is taken and is allowed to play against an agent from 50th generation. We set the rules of the game such that both the agents are allowed to play 20 moves each, and the 50th generation agent always begin playing each game with a same initial random network on which it is trained, whereas the 2000th generation agent continues with the network it had at the end of the previous game. The 2000th generation agent plays as white while the 50th generation agent plays as black. The genotype of the agents is kept the same throughout this experiment, only the architecture develop (number of neurons and neurites change) and shape (neurite branches continue to shrink and grow) during the course of game. At the end of each game we have calculated the fitness of both the agents, and plotted against each other. Figure 3(left) shows the fitness variations of the two agents calculated at the end of every game, and Figure 3(right) the fitness averaged over five consecutive games. The 2000th fitness represented by undashed line is always above zero, showing clearly that its performance is better throughout 500 games, while developing its network, but not that it improves at all. Figure 4(left) shows the accumulated fitness graph of the well evolved agent over 500 games. From these graphs it is evident that although changes in the network occur during every game, the network sustains its integrity of getting higher fitness over the less evolved agent, demonstrating that it does not forget how to play checkers better. However, figure 3(left) shows some peaks at various stages, these are the cases when the highly evolved agent beat the opponent within 20 moves. This is very interesting since while being evolved the agent was

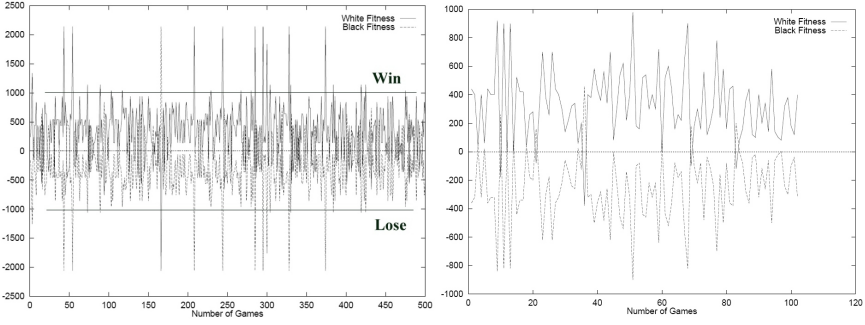


Fig. 3. Fitness variation (left) of a highly evolved agent against less evolved agent, and average fitness over every five games (right)

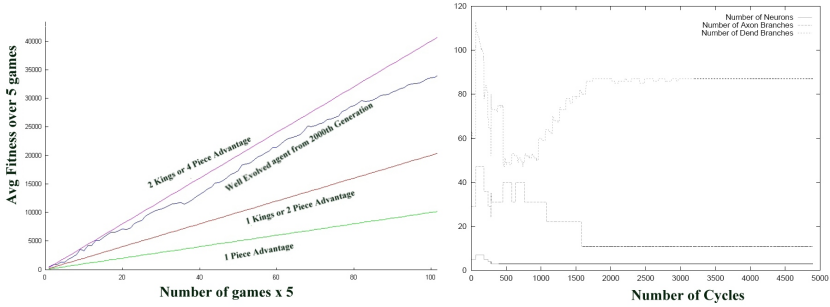


Fig. 4. Graph showing the accumulated fitness (left), and network statistics (right)

never able to beat its opponent within 20 moves, as it is trained against a high skilled checkers program (MCP). But during the development stage when it is allowed to play more than 5 games it is able to do so. As it continues to develop and play without evolution, its ability to beat the opponent within 20 moves seems to increase as evident by the average fitness staying above the x-axis in Figure 3(right). Figure 4(right) shows the variation in number of neurons and neurites of a well evolved agent (2000th generation) during all the games. From the figure it is evident that initially the network changes quite a lot, at some point reducing to a minimum structure, and then stabilizes to a structure with a fixed number of neurons and neurites. This is quite interesting as the network is still allowed to develop but it stabilizes itself, while its branches continue to move around and the weights of neurites continue to update. The network is not trained to find a small network, but when it plays games continuously it continues to change until it finds a minimal suitable structure that can play well. This is evident from the accumulated fitness graph in figure 4 (left). From a close analysis we found that every time a new game is started the network although updated, repeats its initial moves, these cause it to make two double jumps taking four pieces of the opponent at the start. This is interesting behaviour as the opponent always starts with the same initial structure, and it will repeat the

same moves if the developed agent (well evolved) also does. A number of games are studied starting from game 100 (when the network appears to stabilize), the agent seems to repeat its first 8 moves almost every time and this causes the agent to take two double jumps over the opponent giving it an extra advantage. The developed agent does not know when one game ends and another begins, yet it makes the same initial moves with a different network, forcing the opponent to repeat the same mistakes, causing the opponent to lose the game. This suggests that the agent responds to changes in board positions and is able to make the same moves with a different network. This demonstrates that stable behaviour can be obtained when the CGPCN is changing.

4 Conclusion

We have described a neuron-inspired developmental approach to construct a new kind of computational neural architectures which has the potential to learn through experience. We found that the neural structures controlling the agents grow and change in response to their behaviour, interactions with each other and the environment, and allow them to learn and exhibit intelligent behaviour. We used a technique called CGP to encode and evolve seven computational functions inspired by the biological neuron. The eventual aim is to see if it is possible to evolve a network that can learn by experience.

References

1. Marcus, G.: *The Birth of the Mind*. Basic Books (2004)
2. Smythies, J.: *The Dynamic Neuron*. Bradford (2002)
3. Rose, S.: *The Making of Memory: From Molecules to Mind*. Vintage (2003)
4. Koch, C., Segev, I.: The role of single neurons in information processing. *Nature Neuroscience Supplement* 3, 1171–1177 (2000)
5. Kandel, E.R., Schwartz, J.H., Jessell, T.M.: *Principles of Neural Science*, 4th edn. McGraw-Hill, New York (2000)
6. Bestman, J., Santos Da Silva, J., Cline, H.: *Dendrites: Dendrite Development*. Oxford University Press, Oxford (2008)
7. Khan, G.: *Thesis: Evolution of Neuro-inspired developmental programs capable of Learning*. Department of Electronics, University of York (2008)
8. Koza, J.: *Genetic Programming: On the Programming of Computers by Means of Natural selection*. MIT Press, Cambridge (1992)
9. Miller, J.F., Thomson, P.: Cartesian genetic programming. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) *EuroGP 2000*. LNCS, vol. 1802, pp. 121–132. Springer, Heidelberg (2000)
10. Khan, G., Miller, J., Halliday, D.: Coevolution of intelligent agents using cartesian genetic programming. In: *Proc. GECCO*, pp. 269–276 (2007)
11. Shannon, C.: Programming a computer for playing chess. *Phil. Mag.* 41, 256–275 (1950)
12. Dimand, R.W., Dimand, M.A.: *A History of Game Theory: From the Beginnings to 1945*, Routledge, vol. 1 (1996)

A New Approach to Adapting Control Parameters in Differential Evolution Algorithm

Liang Feng, Yin-Fei Yang, and Yu-Xuan Wang

School of Communications and Information Engineering
Nanjing University of Posts and Telecommunications
{brightfengs, yangyin7, logpie}@gmail.com

Abstract. In Differential Evolution, control parameters play important roles in balancing the exploration and exploitation capability, and different control parameters are required for different types of problems. However, finding optimal control parameters for each problem is difficult and not realistic. Hence, we propose a method to adjust them adaptively in this paper. In our proposed method, whether or not the current control parameters will be adjusted is based on a probability that is adaptively calculated according to their previous performance. Besides, normal distribution with variable mean value and standard deviation is employed to generate new control parameters. Performance on a set of benchmark functions indicates that our proposed method converges fast and achieves competitive results.

Keywords: Differential Evolution, Adaptive Parameter Control, Normal Distribution.

1 Introduction

Differential Evolution (DE) is a global optimization algorithm that originally proposed by Price and Storn in 1995 [1]. Due to its simplicity and strong global search capability, it has been studied extensively and widely used in many real-world problems [1,2]. DE only involves three parameters: F (*Scaling Factor*), CR (*Crossover Probability*) and NP (*Population Size*) [3,4]. The performance of DE highly depends on the control parameters F and CR . In general, large F favors global exploration, while small F favors local exploitation [5]. CR determines how many new components can be introduced in a newly generated solution. Storn [6] suggested that the initial choice of the control parameters are: $F \in [0.5, 1]$, $CR \in [0.8, 1]$ and $NP = 10 \times D$ where D is the dimensionality of the problem. The range of F and CR are commonly limited to $[0, 1]$, and the following parameter settings are commonly used : $NP = 100$, $F = 0.5$, and $CR = 0.9$ [1,4,7,8].

DE is sensitive to the choice of control parameters, and it is difficult to find the optimal control parameters for different problems [9]. Some methods have been proposed to adjust control parameters in a self-adaptive or adaptive manner [3,4,10]. In this paper, we propose a method to adaptively adjust control

parameters. Whether or not the current control parameters will be adjusted is based on a probability which is adaptively calculated according to their previous performance. This paper is organized as follows: Section 2 simply introduces the Differential Evolution; related work about self-adaptive or adaptive adjustment of control parameters are discussed in Section 3; our proposed method is presented in Section 4, and experiments are carried out to demonstrate the effectiveness of our proposed method in Section 5; finally, Section 6 concludes this paper.

2 Basic Differential Evolution

Several variants of DE have been proposed in literature [1,2]. For simplicity, only DE/*rand/1/bin* is discussed here. Assume $f(x)$, $\mathbf{x} = [x_1, x_2, \dots, x_D] \in R^D$ is a function needed to be minimized. At the beginning, DE initializes NP individuals randomly in the search space, and then three operations (mutation, crossover and selection) are sequentially performed on each individual until a stopping criterion is satisfied. Let $\mathbf{x}_{i,g}$ denote individual i ($i = 1, \dots, NP$) at iteration g , and denote $x_{i,g}^j$ as the j th ($j = 1, \dots, D$) component of individual i at iteration g .

Mutation: For individual i at generation g , a mutant vector $\mathbf{v}_{i,g+1}$ is generated by mutation as follows:

$$\mathbf{v}_{i,g+1} = \mathbf{x}_{r_1,g} + F \times (\mathbf{x}_{r_2,g} - \mathbf{x}_{r_3,g}) \quad (1)$$

where, r_1, r_2 and r_3 are randomly selected from the whole population, and r_1, r_2, r_3 and i must be mutually different. $\mathbf{x}_{r_1,g}$ is base vector, $(\mathbf{x}_{r_2,g} - \mathbf{x}_{r_3,g})$ is denoted as difference vector, and F is the scaling factor which controls the amplitude of difference vector.

Crossover: For individual i at generation g , a trial vector $\mathbf{u}_{i,g+1}$ is generated by performing a crossover operation on target vector $\mathbf{x}_{i,g}$ and mutant vector $\mathbf{v}_{i,g+1}$ under crossover probability CR .

$$u_{i,g+1}^j = \begin{cases} v_{i,g+1}^j, & \text{if } (rand \leq CR) \text{ or } (j = rand_j) \\ x_{i,g}^j, & \text{otherwise} \end{cases} \quad (2)$$

where $rand$ is uniformly drawn from $[0, 1]$, and $rand_j$ is randomly selected from $[1, \dots, D]$, it is used to ensure that at least one dimension of trial vector is changed.

Selection: For individual i at generation g , a greedy selection mechanism is used to determine whether the trial vector $\mathbf{u}_{i,g+1}$ can survival to the next generation, that is defined as follows:

$$\mathbf{x}_{i,g+1} = \begin{cases} \mathbf{u}_{i,g+1}, & \text{if } f(\mathbf{u}_{i,g+1}) < f(\mathbf{x}_{i,g}) \\ \mathbf{x}_{i,g}, & \text{otherwise} \end{cases} \quad (3)$$

3 Related Work

Various methods have been proposed to change F and CR in a self-adaptive or adaptive manner [3,4,8,10]. They differ in two major aspects. Firstly, they differ in when the parameters are changed. Secondly, they differ in how the new control parameters are generated.

Liu and Lampinen adopted fuzzy logic controllers to adjust control parameters F and CR at each generation (FADE) [4], and the adaptation is performed on population level. Control parameters are adjusted so that the fitness of the successive generations can be minimized.

A self-adaptive Differential Evolution algorithm (SaDE) is proposed by Qin and Suganthan to change the learning strategy and control parameters in a self-adaptive manner [10]. The adaptation of control parameters is performed on individual level, and each individual i is associated with F_i and CR_i . According to their work, F_i is generated at each generation by a normal distribution $N(0.5, 0.3)$. They assume CR_i obeys normal distribution with standard deviation 0.1, and the mean value of CR_i is updated every 5 generations according to successful trial vectors' CR_i in the past 25 generations. More information about how the learning strategy and control parameters are updated can be found in [10].

J. Brest *et. al* proposed a self-adaptive method (jDE) to adjust F_i and CR_i at each generation according to a pre-defined probability Pr_i [3], and Pr_i is set to be 0.1 for all individuals, that means F_i and CR_i will be changed every 10 generations in average. Besides, uniform distribution is adopted to update control parameters. F_i and CR_i are uniformly drawn from range $[0.1, 1]$ and $[0, 1]$ respectively.

4 Proposed Method

In this section, we propose a new adaptive method to adjust F_i and CR_i simultaneously because the performance of DE is jointly affected by them. Firstly, a method is proposed to determine when control parameters should be adjusted. Secondly, a discussion is made on how to generate new control parameters.

4.1 When to Adjust Control Parameters

A question should be answered first: why do we need to adjust control parameters? A good answer is that current control parameters are no longer suitable for current individual as the evolution proceeds. Recall that jDE [3] adjusts control parameters according to a pre-defined probability Pr_i no matter whether current control parameters are suitable or not. Thus, both suitable and potential suitable control parameters cannot be preserved effectively, and unsuitable control parameters cannot be adjusted in time.

We argue that probability Pr_i should be adaptively adjusted according to the suitability of current control parameters. Now a second question should be answered: can we provide a simple method to measure the suitability of current control parameters? We believe that it is the suitable control parameters used

on the target vector that generate better trial vector. That is to say, the fitness of the target vector and trial vector can provide us some information about the suitability of current control parameters. Specifically, if the trial vector is better than the target vector, we consider current control parameters are suitable and should not be adjusted; if the trial vector is marginally worse than the target vector, we regard current control parameters as potential suitable parameters and should not be adjusted too quickly; and if the trial vector is much worse than the target vector, we consider current control parameters are unsuitable and should be adjusted immediately.

Inspired from Simulated Annealing (SA) [11], the following rule is used to adaptively adjust probability Pr_i :

$$Pr_i = 1 - \exp(-\beta(f_{trial} - f_{target})) \quad (4)$$

where, Pr_i denotes the probability of changing current control parameters, f_{trial} and f_{target} are the fitness of the trial vector and target vector, respectively. β is a parameter which is used to control the amplitude of $f_{trial} - f_{target}$ on probability Pr_i . For simplicity, $\beta = 1/\max(f_{trial}^i)$ ($i = 1, \dots, NP$) is used in this paper. If $f_{trial} < f_{target}$, then $Pr_i < 0$, so current control parameters will not be adjusted because they are suitable. If $f_{trial} \geq f_{target}$, then $Pr_i \geq 0$, so current control parameters will be changed based on probability Pr_i . Besides, the larger Pr_i is, the more frequently current control parameters are adjusted. For example, if $Pr_i = 0.1$, it will take 10 generations to adjust current control parameters in average, thus potential suitable control parameters can be preserved; and if $Pr_i = 0.5$, it will take 2 generations to adjust current control parameters in average, thus unsuitable control parameters can be replaced in time.

4.2 How to Generate Control Parameters

To generate new control parameters, uniform distribution is used in jDE [8] and normal distribution is adopted in SaDE [10]. However, the individuals' information is not fully exploited to generate new control parameters. Since the difference between f_{trial} and f_{target} indicates how suitable current control parameters are, then it is reasonable to generate new control parameters by exploiting this information. If an individual finds suitable control parameters, it is better to share it with other individuals to help them find better solutions. Denote $F_{b,g}$ and $CR_{b,g}$ as the best target vector's control parameters at generation g . Then, some individuals' $F'_{i,g}$ and $CR'_{i,g}$ will be generated around $F_{b,g}$ and $CR_{b,g}$, respectively. To generate a series of random numbers centered at a specific value, normal distribution would be a better choice than uniform distribution.

Since normal distribution is adopted to generate $F'_{i,g}$ and $CR'_{i,g}$, the next step is to determine the standard deviation for each individual. Let $F_{m,g}$ and $CR_{m,g}$ be the mean values of all the individuals' control parameters at generation g . For a better control of the diversity of all the individual's control parameters, the distance between $F_{i,g}$ (or $CR_{i,g}$) and $F_{m,g}$ (or $CR_{m,g}$) are scaled and then used as standard deviation, that's $\gamma|F_{m,g} - F_{i,g}|$ (or $\gamma|CR_{m,g} - CR_{i,g}|$). If $\gamma > 1$, the diversity of control parameters tends to increase; if $\gamma < 1$, the diversity tends to

decrease; and if $\gamma = 1$, the diversity can be preserved. For simplicity, $\gamma = 1$ is used in this paper.

4.3 Updating Rules of Control Parameters

In our proposed method, the initial $F_{i,1}$ and $CR_{i,1}$ are generated uniformly over $[0, 1]$. After sequential execution of mutation, crossover and selection on individual i at generation g , $F_{i,g}$ and $CR_{i,g}$ of individual i are adjusted according to the following rules:

$$F'_{i,g} = \begin{cases} \mathcal{N}(F_{b,g}, \gamma |F_{m,g} - F_{i,g}|^2), & \text{if } rand_1 \leq Pr_i \\ F_{i,g}, & \text{otherwise} \end{cases} \quad (5)$$

$$CR'_{i,g} = \begin{cases} \mathcal{N}(CR_{b,g}, \gamma |CR_{m,g} - CR_{i,g}|^2), & \text{if } rand_2 \leq Pr_i \\ CR_{i,g}, & \text{otherwise} \end{cases} \quad (6)$$

where, $rand_1$ and $rand_2$ are random numbers uniformly drawn from $[0, 1]$, Pr_i determines whether individual i 's control parameters should be adjusted, $F_{i,g}$ and $CR_{i,g}$ are control parameters of individual i at generation g , respectively. $F_{b,g}$ and $CR_{b,g}$ are the best target vector's control parameters at generation g , and $F_{m,g}$ and $CR_{m,g}$ are mean values of all the individuals' control parameters at generation g .

5 Experiment

5.1 Experimental Settings

To demonstrate the effectiveness of our proposed method, DE [2], FADE [4], jDE [3] and SaDE [10] are used for comparison. They are tested on the benchmark functions listed in Table 1 with dimensionality of 30 and they are to be minimized. All the five algorithms are independently executed 30 times with maximum generation $maxGen = 2000$ and population size $NP = 100$.

The scaling factor and crossover probability used for DE are $F = 0.5$ and $CR = 0.9$ [8], respectively. The configurations of FADE, jDE and SaDE are the same with their corresponding literature.

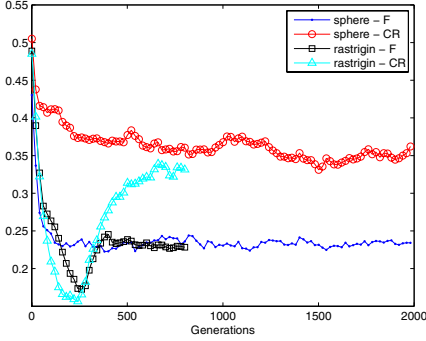
5.2 Results

The results of all the five algorithms are listed in Table 2. In general, our proposed method cannot perform well consistently on all the unimodal functions, but it can achieve competitive results on multimodal functions. Our proposed method performs very well on Sphere function, but it cannot obtain a satisfactory result on Rosenbrock function. On Quadric function, it is surpassed by SaDE in the later stage.

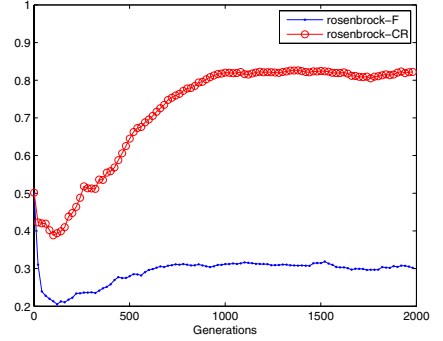
The convergence graph of some representative functions are plotted in Figure 2. Our proposed method converges slightly faster at the early stage on

Table 1. Benchmark Functions

	Test Function	Range
Sphere	$f_1(x) = \sum_{i=1}^D x_i^2$	$[-100, 100]$
Quadric	$f_2(x) = \sum_{i=1}^D (\sum_{j=1}^i x_j)^2$	$[-100, 100]$
Schwefel2.22	$f_3(x) = \sum_{i=1}^D x_i + \prod_{i=1}^D x_i $	$[-10, 10]$
Rosenbrock	$f_4(x) = \sum_{i=1}^{D-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	$[-30, 30]$
Rastrigin	$f_5(x) = \sum_{i=1}^D [x_i^2 - 10 \cos(2\pi x_i) + 10]$	$[-5.12, 5.12]$
Griewank	$f_6(x) = \frac{1}{4000} \sum_{i=1}^D x_i^2 - \prod_{i=1}^D \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-600, 600]$
Ackely	$f_7(x) = -20 \exp\left(-0.2 \sqrt{\frac{1}{D} \sum_{i=1}^D x_i^2}\right) - \exp\left(\frac{1}{D} \sum_{i=1}^D \cos(2\pi x_i)\right) + 20 + e$	$[-32, 32]$
Schwefel	$f_8(x) = 418.9829 \times D - \sum_{i=1}^D (x_i \sin(\sqrt{ x_i }))$	$[-500, 500]$
Noise	$f_9(x) = \sum_{i=1}^D i x_i^4 + \text{rand}[0, 1)$	$[-1.28, 1.28]$
Penalized2	$f_{10}(x) = 0.1 \{\sin^2(3\pi x_1) + \sum_{i=1}^{D-1} (x_i - 1)^2 [1 + \sin^2(3\pi x_{i+1})] + (x_D - 1)^2 [1 + \sin^2(2\pi x_D)]\} + \sum_{i=1}^D u(x_i, 5, 100, 4)$	$[-50, 50]$



(a) Sphere and Rastrigin



(b) Rosenbrock

Fig. 1. Mean value of whole population's control parameters over generation on Sphere, Rastrigin and Rosenbrock

unimodal function. On multimodal functions, it converges much faster on Rastrigin and Schwefel function, and it converges marginally faster on Griewank and penalized2 functions. On Noise function, the convergence speed of all the five algorithms are almost the same with each other.

To see how our proposed adaptive method works, the mean value of all individual's $F_{i,g}$ and $CR_{i,g}$ are recorded at each generation and plotted in Figure 1 for some representative functions.

5.3 Results Analysis

We will analyze the performance of our proposed method based on Figure 1. On Sphere function, small control parameters are found to facilitate local exploitation in the early stage, and suitable control parameters found in the early stage are kept almost unchanged in the later stage, thus few function evaluations are

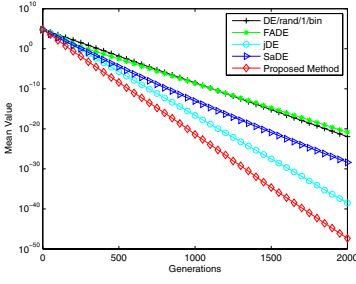
Table 2. Performance comparison among jDE, SaDE, FADE, DE/rand/1/bin and our proposed method on benchmark functions

	Proposed Method Mean (Std Dev)	jDE Mean (Std Dev)	SaDE Mean (Std Dev)	FADE Mean (Std Dev)	DE/rand/1/bin Mean (Std Dev)
f_1	4.50×10^{-48} (9.18×10^{-48})	3.13×10^{-39} (4.04×10^{-39})	4.17×10^{-29} (6.14×10^{-29})	1.40×10^{-21} (6.48×10^{-22})	1.10×10^{-22} (1.00×10^{22})
f_2	3.23×10^{-5} (2.74×10^{-5})	8.49×10^{-4} (9.75×10^{-4})	2.63×10^{-9} (7.90×10^{-9})	1.29×10^4 (2.38×10^3)	3.23×10^{-2} (2.28×10^{-2})
f_3	5.14×10^{-30} (8.37×10^{-30})	1.58×10^{-23} (1.17×10^{-23})	1.17×10^{-15} (6.04×10^{-16})	2.10×10^{-12} (6.01×10^{-13})	5.03×10^{-11} (3.68×10^{-11})
f_4	22.25 (22.11)	20.14 (14.66)	3.95 (4.79)	0.41 (0.32)	9.79 (1.09)
f_5	0 (0)	0 (0)	1.00×10^{-15} (2.62×10^{-15})	89.83 (10.05)	1.62×10^2 (1.59×10^1)
f_6	0 0	0 0	0 0	0 0	0 0
f_7	5.38×10^{-15} (1.59×10^{-15})	3.55×10^{-15} (0)	8.94×10^{-15} (2.07×10^{-15})	3.49×10^{-11} (6.89×10^{12})	2.52×10^{-12} (1.16×10^{-12})
f_8	0 0	0 0	0 0	0 0	5.72×10^3 (6.14×10^2)
f_9	5.00×10^{-3} (1.10×10^{-3})	5.00×10^{-3} (1.20×10^{-3})	4.80×10^{-3} (1.40×10^{-3})	1.21×10^{-2} (2.90×10^{-3})	7.40×10^{-3} (2.10×10^{-3})
f_{10}	1.34×10^{-32} (5.56×10^{-48})	9.23×10^{-32} (5.56×10^{-47})	1.17×10^{-27} (1.86×10^{-27})	6.54×10^{-22} (3.36×10^{-22})	8.15×10^{-20} (1.31×10^{-19})

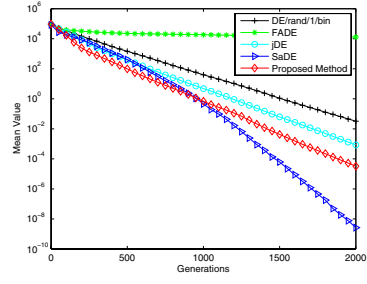
spent on finding suitable control parameters and it can converge fast and achieve good result.

On Rastrigin function, from Figure 1(a), we can find that small control parameters are favored for this separable function to locate the region of global optima in the early stage. Besides, due to the existence of many local minima, a relatively high diversity of control parameters are preserved by the whole population. When the whole population approaches the region of global optima in the later stage, the landscape changes from complex to simple. The whole population can adapt to this change and new suitable control parameters can be found very fast, and the mean value of the whole population's $F_{i,g}$ and $CR_{i,g}$ on Sphere and Rastrigin are almost the same with each other in the later stage.

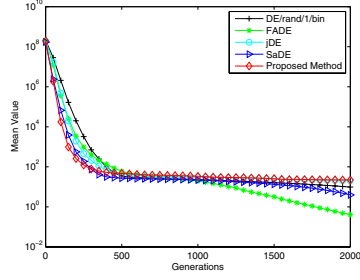
Focus on Rosenbrock, due to the relatively simple landscape for the function outside the valley, small F and CR are favored to quickly exploiting the space, so from Figure 1(b), we can find the control parameters tend to small values quickly, and then the diversity is gradually lost. But landscape changes from simple to complex when the population approaches the narrow valley in the later stage, the whole population can detect it and CR increase quickly, but due to the loss of diversity of control parameters in the early stage, it takes lots of generations to find new suitable control parameters. If more function evaluations



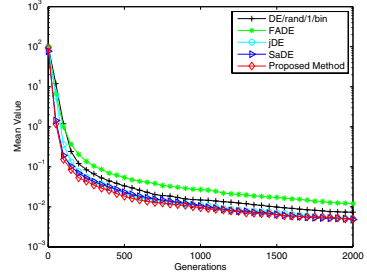
(a) Sphere



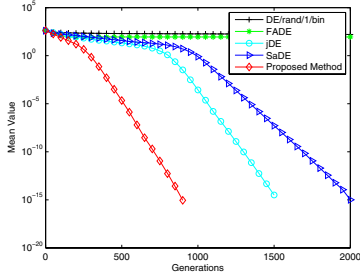
(b) Quadric



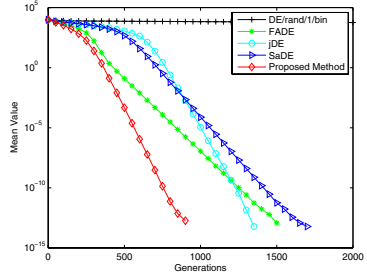
(c) Rosenbrock



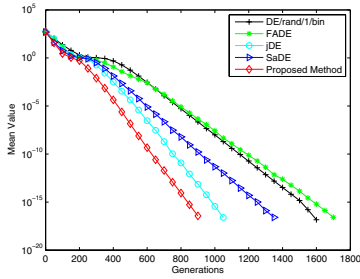
(d) Noise



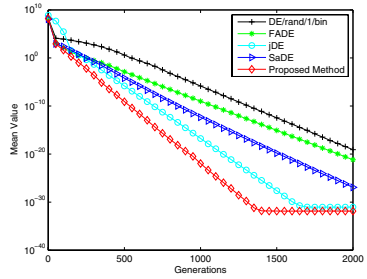
(e) Rastrigin



(f) Schwefel



(g) Griewank



(h) Penalized2

Fig. 2. Convergence graph of DE/rand/1/bin, FADE, jDE, SaDE and our proposed method on some representative benchmark functions

are given, the whole population can finally adapt to the change of landscape and find the global optima.

6 Conclusions

In this paper, we proposed a method to adjust the control parameters adaptively. Whether the current control parameters of individual i will be adjusted is controlled by a probability Pr_i . For the purpose of preserving potential suitable control parameters and replacing unsuitable control parameters in time, we introduced a method to adaptively adjust probability Pr_i based on the performance of control parameters. Regarding how the new control parameters are generated, we introduced a greedy mechanism to adjust unsuitable control parameters based on suitable control parameters of the best individual. Performance on a set of functions indicates that our proposed method can achieve fast convergence speed and competitive results. But our method may not be good at dealing with the change of landscape from simple to complex effectively, and our primary analysis shows that loss of diversity of control parameters is one reason for this phenomena. Therefore, an extension of this work is to find better methods to deal with the change of landscape from simple to complex by increasing the diversity of control parameters, and three methods are worthy to be investigated in the future. Firstly, smaller β can be used to decrease the probability Pr_i so that more potential suitable control parameters can be preserved. Secondly, a wide range of control parameters can be generated by using larger γ . Thirdly, introduce a method to detect the change of landscape based on improvement of the whole population's fitness, and then reinitialize control parameters $F_{i,g}$ and $CR_{i,g}$ to obtain a high diversity of control parameters.

Acknowledgment

The authors would like to thank Professor Suganthan for providing the codes of SaDE and Professor J.Brest for providing the codes of jDE.

References

1. Storn, R., Price, K.: Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Internation Computer Science Institute, Berkley, Tech. Rep. (1995)
2. Storn, R., Price, K.: Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11, 341–359 (1997)
3. Brest, J., Greiner, S., Bošković, B., Mernik, M., Žumer, V.: Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. IEEE Transactions on Evolutionary Computation 10, 646–657 (2006)
4. Liu, J., Lampinen, J.: A fuzzy adaptive differential evolution algorithm. In: TENCON 2002. Proceedings. 2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering, vol. 1, pp. 606–611 (October 2002)

5. Das, S., Konar, A., Chakraborty, U.K.: Two improved differential evolution schemes for faster global search. In: GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation, pp. 991–998 (2005)
6. Storn, R.: On the usage of differential evolution for function optimization. In: Fuzzy Information Processing Society, 1996. NAFIPS. Biennial Conference of the North American, June 1996, pp. 519–523 (1996)
7. Ali, M.M., Törn, A.: Population set-based global optimization algorithms: some modifications and numerical studies. *Comput. Oper. Res.* 31, 1703–1725 (2004)
8. Brest, J., Bošković, B., Greiner, S., Žumer, V., Maučec, M.S.: Performance comparison of self-adaptive and adaptive differential evolution algorithms. *Soft Comput.* 11, 617–629 (2007)
9. Liu, J., Lampinen, J.: On setting the control parameters of differential evolution method. In: *Proc. 8th Int. Conf. Soft Computing*, pp. 11–18 (2002)
10. Qin, A.K., Suganthan, P.N.: Self-adaptive differential evolution algorithm for numerical optimization. In: *The 2005 IEEE Congress on Evolutionary Computation 2*, pp. 1785–1791 (2005)
11. Kirkpatrick Jr., S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. *Science* 220, 671–680 (1983)

A Novel Genetic Algorithm with Orthogonal Prediction for Global Numerical Optimization

Jun Zhang*, Jing-Hui Zhong, and Xiao-Min Hu

Department of Computer Science, SUN Yat-sen University,
510275 Guangzhou, China
junzhang@ieee.org

Abstract. This paper proposes a novel orthogonal predictive local search (OPLS) to enhance the performance of the conventional genetic algorithms. OPLS operation predicts the most promising direction for the individuals to explore their neighborhood. It uses the orthogonal design method to sample orthogonal combinations to make the prediction. The resulting algorithm is termed the orthogonal predictive genetic algorithm (OPGA). OPGA has been tested on eleven numerical optimization functions in comparison with some typical algorithms. The results demonstrate the effectiveness of the proposed algorithm for achieving better solutions with a faster convergence speed.

Keywords: Genetic algorithm, orthogonal design method, local search, evolutionary algorithm, numerical optimization.

1 Introduction

Genetic algorithms (GAs) [1] are search techniques which simulate the biological evolution in the nature, using genetic operations such as selection, crossover (recombination), and mutation to train the population of individuals (chromosomes or candidate solutions) from generation to generation. Individuals with high fitness values have higher probability to survive in the next generation. As GAs do not require extensive knowledge of the search space, such as functional derivatives, to search for the optimal solution of the problem instances, they have been widely applied in engineering [2][3], scheduling [4][5], network and routing design [6][7], and numerical optimization [8][9], etc. However, the convergence speed of the classical GA (CGA) is slow and the solution quality is undesirable.

In order to enhance the performance of GAs, various methods have been proposed. Instead of using the traditional binary encoding, the real encoding method has been widely used in global numerical optimization to obtain solutions with higher precision. Different selection, crossover, and mutation operators have also been proposed. Adding a local search operation to GAs is a straightforward way to introduce local refinement processes into a GA cycle. The Gaussian mutation is a well-known local search operator for choosing a different point in the neighborhood of a solution. Yao and Liu [9] further improved the operator by using the Cauchy mutation to replace the

* Corresponding author.

Gaussian mutation. The basic idea of local search is to explore the neighborhood of solutions for better results. However, most of the existing local search operations are stochastic and lack an effective method to guide the search.

This paper proposes a novel orthogonal prediction method in the local search operation of GA. The resulting algorithm is termed the orthogonal predictive genetic algorithm (OPGA). The orthogonal predictive local search (OPLS) is based on the orthogonal design method [10][11] to predict the promising direction for moving the individual. By considering each variable in the numerical optimization problem as a factor, and the movements of the candidate solution as levels, the orthogonal design method selects some orthogonal points in the neighborhood of the individual according to an orthogonal array. Although the number of the orthogonal points is small, these points locate orthogonally with each other, reflecting the optimization condition of the neighborhood. After the orthogonal points are evaluated, the importance of different levels of variables is analyzed. The combination of the most important levels of each variable represents the predicted best direction for the individual to explore further. This paper is different from the previous work in [12], which did not have the prediction step and only used the orthogonal array to find a delegate for representing the surrounding regions.

Using the orthogonal design method to predict the potential enhancement for an individual is groundbreaking. First, the orthogonal design method is an effective multifactor experimental method, which finds a high-quality multifactor combination with small effort. Second, the prediction-based local search not only reduces the blindness of the traditional local search, but also provides more chances for the quick approach to the global optimum. Third, the proposed prediction strategy has addressed the common problem that exists in most of the numerical optimization algorithms. It has wide future applications in the other optimization fields.

Numerical experiments have been made to evaluate the performance of the proposed algorithm in solving some benchmark unimodal and multimodal numerical functions. The rest of the paper is organized as follows. Section 2 defines the numerical optimization and then briefly introduces the concepts of the orthogonal design method. Section 3 presents the proposed orthogonal predictive genetic algorithm. The numerical experiments are made in Section 4. Finally, Section 5 draws the conclusion.

2 Preliminary

This paper proposed a novel genetic algorithm to solve global numerical optimization problems. Here the optimization problem is formally defined. Then a brief introduction to the orthogonal design method is presented.

2.1 Problem Definition

The formal definition of the global numerical optimization problem is as follows

$$\begin{aligned} & \text{maximize } f(\mathbf{x}) \\ & \text{subject to } \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \end{aligned} \quad (1)$$

where $\mathbf{x} = (x_1, x_2, \dots, x_n)$ is the variable vector of the problem, n is the number of variables, $\mathbf{l} = (l_1, l_2, \dots, l_n)$ is the lower boundary vector of the variables, and $\mathbf{u} = (u_1, u_2, \dots, u_n)$ is the upper boundary vector of the variables. The maximization problem is to find

the maximum result in the solution space. If the problem is a minimization one, we can use $-f(x)$ to change it into the equivalent maximization problem.

2.2 Orthogonal Design Method

The orthogonal design method [10][11] is one of the most widely used experimental design methods in agricultural experiments, science research, manufacturing industries and quality management, etc. It provides an efficient way to find a good sample for the experiment with multifactors. Each factor in the experiment has some test values which are called levels.

Generally, for an experiment with k factors, if each factor has s levels, the total number of combinations is s^k . To test the entire s^k combinations may be prohibitively infeasible when the evaluation of each combination costs heavily and the value of k or s is large. The orthogonal design method can be used to achieve a high-quality combination with a much smaller number of tests. It uses an orthogonal array to design the experiment. We use $OA(N, k, s)$ to denote an orthogonal array with N rows, k factors, and s levels for each factor. The value N is the number of combinations to be tested in the experiment. The method for generating an orthogonal array is shown in the Appendix. The orthogonal design method is composed of two steps.

■ **Step 1:** Use an orthogonal array to evaluate the orthogonal combinations.

Here is an example of using an orthogonal array $OA(9, 4, 3)$, which is illustrated in Table 1. Suppose there are three factors x_1, x_2, x_3 to be tested. According to the first three columns in Table 1, each combination can be presented as a node in Fig. 1. The three dimensional coordinates of the cube stand for the three factors. Each factor has three levels, which are symbolized as ' $F^{(i)}$ ', ' $S^{(i)}$ ', ' $B^{(i)}$ ', where $i=1,2,3$ are the indexes of the factors. In a full-scale experiment, there are $3^3=27$ combinations to be tested, whereas the orthogonal design method reduces the number of combinations to be only nine. As each edge and each face of the cube have the same number of orthogonal points respectively, these nine points distribute uniformly and orthogonally in the cube.

■ **Step 2:** Generate the orthogonal design combination.

Using the first three columns of the orthogonal array in Table 1, the experiment can be implemented as the upper part of Table 2. Each orthogonal combination is evaluated by f , the results are shown in the fifth column of Table 2. For each factor, the sums of evaluation values to each of the three levels are calculated and the level with the largest

Table 1. An orthogonal array $OA(9, 4, 3)$

Factor Combination	1	2	3	4
C_1	1	1	1	1
C_2	1	2	2	2
C_3	1	3	3	3
C_4	2	1	2	3
C_5	2	2	3	1
C_6	2	3	1	2
C_7	3	1	3	2
C_8	3	2	1	3
C_9	3	3	2	1

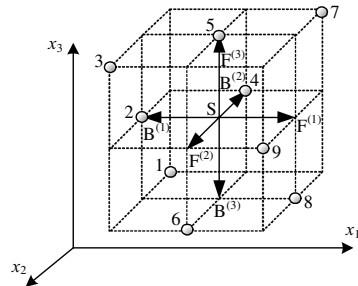


Fig. 1. The orthogonal illustration according to the first three columns in Table 1. The level 1 is moving backward (B). The level 2 is keeping stationary (S). The level 3 is moving forward (F).

Table 2. The orthogonal design plan

Factor Combination	x_1	x_2	x_3	Evaluation Value
C_1	1	1	1	$f^{(1)}=5$
C_2	1	2	2	$f^{(2)}=3$
C_3	1	3	3	$f^{(3)}=2$
C_4	2	1	2	$f^{(4)}=1$
C_5	2	2	3	$f^{(5)}=0.7$
C_6	2	3	1	$f^{(6)}=6$
C_7	3	1	3	$f^{(7)}=1$
C_8	3	2	1	$f^{(8)}=3$
C_9	3	3	2	$f^{(9)}=5$
Sums of Evaluations				
L_1	$f^{(1)}+f^{(2)}+f^{(3)}=10$	$f^{(1)}+f^{(4)}+f^{(7)}=7$	$f^{(1)}+f^{(6)}+f^{(8)}=14$	
L_2	$f^{(4)}+f^{(5)}+f^{(6)}=7.7$	$f^{(2)}+f^{(5)}+f^{(8)}=6.7$	$f^{(2)}+f^{(4)}+f^{(9)}=9$	
L_3	$f^{(7)}+f^{(8)}+f^{(9)}=9$	$f^{(3)}+f^{(6)}+f^{(9)}=13$	$f^{(3)}+f^{(5)}+f^{(7)}=3.7$	
Orthogonal Design Result D	1	3	1	$f(x^{(D)})=9$

sum is selected to generate a new combination. For example, to x_1 , the sums of evaluation values of levels 1, 2, and 3 are 10, 7.7, and 9 respectively. The level 1 is chosen because it has the largest sum. The evaluation value of the new combination $D=(1, 3, 1)$ is compared with the nine orthogonal combinations. The combination with the maximum evaluation value is chosen as the final result of the orthogonal experiment.

In the above example, the fourth factor in $OA(9, 4, 3)$ is not used. Moreover, if we rotate the variables from x_1, x_2, x_3 to x_3, x_1, x_2 , the orthogonal combinations may be different. These nine combinations are still orthogonal with each other in the solution space. Therefore, different columns in the orthogonal table can be chosen in the experiment. Although the best combination in these sampled experiments may not be the best one in the full-scale experiment, it is always a high-quality solution.

3 Orthogonal Predictive Genetic Algorithm

The orthogonal design method can be used to predict the most promising direction for GA to explore. In this section, the proposed orthogonal predictive genetic algorithm (OPGA) is presented. We firstly describe the orthogonal predictive local search (OPLS), including the movement prediction and the realization of the moving action. The overall implementation of OPGA is finally presented.

3.1 The Orthogonal Predictive Local Search

According to the orthogonal design method, we can regard each variable in the problem as a factor. To search for a better local solution, we attempt to slightly move a selected individual from its original location to see if there is any improvement. To a

real-coded individual $\mathbf{x}=(x_1, x_2, \dots, x_n)$ in the GA, such a small movement is realized by increasing or decreasing the values of variables. So we set the levels for each variable to be the three movements – moving forward (F), keeping stationary (S), and moving backward (B). The $OA(N, k, s)$ used here must satisfy $s = 3$ and $k \geq n$.

Step 1: Randomly choose n columns from an $OA(N, k, s)$, and record the indexes of the columns as $q_j, j=1, 2, \dots, n$. Mind that the columns must not be duplicated.

Step 2: According to an orthogonal array $OA(N, k, s)$, the orthogonal points $\mathbf{x}^{(c)}=(x_1^{(c)}, x_2^{(c)}, \dots, x_n^{(c)})$ ($c=1, 2, \dots, N$) for the current individual $\mathbf{x}=(x_1, x_2, \dots, x_n)$ are generated as

$$x_j^{(c)} = x_j + \left[\frac{2(A_{cq_j} - 1)}{s - 1} - 1 \right] \delta_j \quad (2)$$

where $A_{cq_j} \in \{1, 2, 3\}$ stands for the level in row c and column q_j of the orthogonal array, δ_j is the movement length, $j=1, 2, \dots, n$. If $x_j^{(c)}$ violates the domain $[l_j, u_j]$, the value of $x_j^{(c)}$ is reset to the nearest boundary value.

Step 3: Evaluate the fitness value of the orthogonal neighboring points $\mathbf{x}^{(c)}$ ($c=1, 2, \dots, N$).

Step 4: Calculate the sums of fitness value for each level of each factor.

Step 5: Select the levels with the largest sums of fitness values to form a new orthogonal combination $\mathbf{D}$. A new point $\mathbf{x}^{(D)}=(x_1^{(D)}, x_2^{(D)}, \dots, x_n^{(D)})$ is then generated as

$$x_j^{(D)} = x_j + \left[\frac{2(D_j - 1)}{s - 1} - 1 \right] \delta_j \quad (3)$$

where $D_j \in \{1, 2, 3\}$, $j=1, 2, \dots, n$.

Step 6: Evaluate the fitness value of $\mathbf{x}^{(D)}$. Compare its fitness value with the orthogonal point $\mathbf{x}^{(c)}$ ($c=1, 2, \dots, N$). If $f(\mathbf{x}^{(D)})$ is bigger than all $f(\mathbf{x}^{(c)})$ ($c=1, 2, \dots, N$), go to *Step 7*. Otherwise, go to *Step 8*.

Step 7: The orthogonal combination $\mathbf{D}$ is the predicted promising moving direction. One more $\hat{\mathbf{x}}^{(D)}$ is generated as (4) to explore the neighborhood in the direction further.

$$\hat{x}_j^{(D)} = x_j + \left[\frac{2(D_j - 1)}{s - 1} - 1 \right] (\delta_j + \delta_j \cdot \text{rand}_j) \quad (4)$$

where rand_j is a uniform random value in $[0, 1]$, $j=1, 2, \dots, n$.

Step 8: The best result in $\{\mathbf{x}^{(D)}, \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}, \hat{\mathbf{x}}^{(D)}\}$ is recorded as $\mathbf{x}^{(E)}$.

Step 9: If $\mathbf{x}^{(E)}$ is better than the original individual $\mathbf{x}$, use $\mathbf{x}^{(E)}$ to replace the value of $\mathbf{x}$ and extend the movement length as (5). Otherwise reduce the movement length as (6).

$$\delta_j \leftarrow \delta_j v_e \quad (5)$$

$$\delta_j \leftarrow \delta_j v_s \quad (6)$$

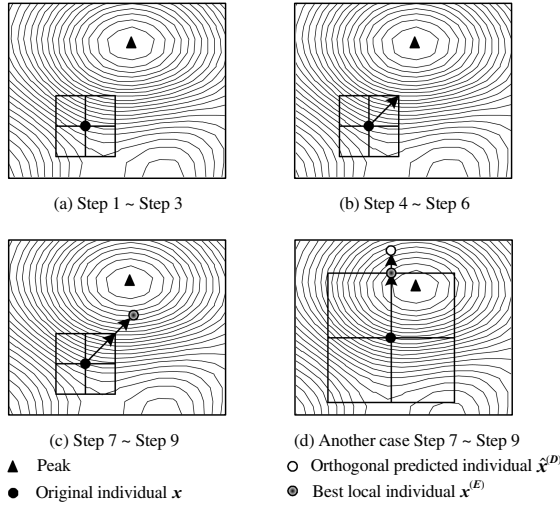


Fig. 2. Illustration of the orthogonal predictive local search (OPLS)

where $v_e \geq 1$ is the expansion rate of the movement length, $0 < v_s \leq 1$ is the shrinking rate of the movement length.

Fig. 2 illustrates the procedure of OPLS. Step 1 to Step 3 use the orthogonal array to sample the orthogonal points. Step 4 to Step 6 generate the orthogonal predictive moving direction. Step 7 to Step 9 explore the direction and update the individual and the movement length. Note that the best local individual is the orthogonal predicted individual in Fig. 2 (c). Fig. 2 (d) presents a case when the movement length is too big to find a better point in the direction.

3.2 Implementation of the Proposed OPGA

The genetic algorithm in this paper follows the basic structure of the classical genetic algorithm. The flowchart of the proposed OPGA is shown in Fig. 3.

Step 1: Initialize the parameters. Randomly generate m individuals in the solution space.

Step 2: Evaluate the individuals with the fitness function. Record the best individual and the corresponding fitness value. Set the *generation*=1.

Step 3: If the termination condition is not met, go to *Step 4*. Otherwise, the algorithm is terminated.

Step 4: Select a new population using a roulette wheel selection method.

Step 5: Perform crossover on the individuals with the crossover rate p_x .

Step 6: Perform mutation on the genes of individuals with the mutation rate p_m .

Step 7: Evaluate the individuals with the fitness function.

Step 8: If the best individual generated in the current generation is better than the recorded best-so-far individual, perform OPLS to the best individual and then update the best-so-far individual. Otherwise, perform OPLS to the recorded best-so-far individual and then use the new best-so-far individual to replace the worst individual in the population.

Step 9: *generation* = *generation* + 1. Go to *Step 3*.

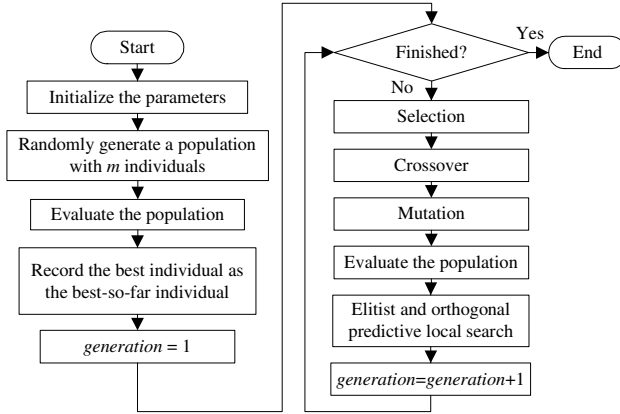


Fig. 3. Flowchart of the proposed orthogonal predictive genetic algorithm (OPGA)

Table 3. A list of the numerical functions
(Domain=variable ranges, Minimum=minimal function value)

Test Functions	Domain	Minimum
$F1 = \sum_{i=1}^{30} x_i^2$	$[-100, 100]^{30}$	0
$F2 = \sum_{i=1}^{30} x_i + \prod_{i=1}^{30} x_i $	$[-10, 10]^{30}$	0
$F3 = \sum_{i=1}^{30} (\sum_{j=1}^i x_j)^2$	$[-100, 100]^{30}$	0
$F4 = \sum_{i=1}^{30} (\lfloor x_i + 0.5 \rfloor)^2$	$[-100, 100]^{30}$	0
$F5 = \sum_{i=1}^{30} -x_i \sin(\sqrt{ x_i })$	$[-500, 500]^{30}$	-12569.5
$F6 = \sum_{i=1}^{30} [x_i^2 - 10 \cos(2\pi x_i) + 10]$	$[-5.12, 5.12]^{30}$	0
$F7 = -20 \exp(-0.2 \sqrt{\frac{1}{30} \sum_{i=1}^{30} x_i^2}) - \exp(\frac{1}{30} \sum_{i=1}^{30} \cos(2\pi x_i)) + 20 + \exp(1)$	$[-100, 100]^{30}$	0
$F8 = \frac{1}{4000} \sum_{i=1}^{30} x_i^2 - \prod_{i=1}^{30} \cos(\frac{x_i}{\sqrt{i}}) + 1$	$[-600, 600]^{30}$	0
$F9 = \frac{\pi}{30} [10 \sin^2(\pi y_1) + \sum_{i=1}^{29} (y_i - 1)^2 [1 + \sin^2(\pi y_{i+1})] + (y_{30} - 1)^2]$ $+ \sum_{i=1}^{30} u(x_i, 10, 100, 4)$ $y_i = 1 + (x_i + 1) / 4, \quad i = 1, 2, \dots, 30$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a \\ 0, & -a \leq x_i \leq a \\ k(-x_i - a)^m, & x_i < -a \end{cases}$	$[-50, 50]^{30}$	0
$F10 = \frac{1}{10} [10 \sin^2(3\pi x_1) + \sum_{i=1}^{29} (x_i - 1)^2 [1 + \sin^2(3\pi x_{i+1})] + (x_{30} - 1)^2 [1 + \sin^2(2\pi x_{30})]]$ $+ \sum_{i=1}^{30} u(x_i, 5, 100, 4)$	$[-50, 50]^{30}$	0
$F11 = \sum_{i=1}^{11} [a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}]^2$ $a_i = \{0.1957 \ 0.1947 \ 0.1735 \ 0.16 \ 0.0844 \ 0.0627 \ 0.0456 \ 0.0342 \ 0.0323 \ 0.0235 \ 0.0246\}$ $b_i^{-1} = \{ \begin{matrix} 0.25 & 0.5 & 1 & 2 & 4 & 6 & 8 & 10 & 12 & 14 & 16 \end{matrix} \}$	$[-5, 5]^4$	$3.075e^{-4}$

4 Numerical Experiments

Eleven numerical functions are tested in this paper, including unimodal and multimodal functions. Table 3 lists the definitions of the functions. OPGA is compared with CGA and the fast evolutionary programming (FEP) [9]. The population size used in OPGA and CGA is $m=20$. The crossover rate is $p_x=0.7$, whereas the mutation rate is $p_m=0.07$. The initial movement length δ_j is $(u_j - l_j)/m/2, j=1,2,\dots,n$. The shrinking rate of the movement length is $v_s=0.5$ for F1, F2, F3, F4, F6, F8, and F9, whereas $v_s=0.9$ for F5, F7, F10, and F11. We have $v_c=1/v_s$ in the experiment. The parameters settings of FEP can be referred to [9].

Table 4. Means and standard deviations. All results were averaged over 100 runs.

F	maxE	CGA		FEP		OPGA	
		mean	St. dev	mean	St. dev	mean	St. dev
1	150000	3.88E-01	1.62E-01	4.39E-04	6.01E-05	1.48E-288	0
2	150000	2.89E-01	5.17E-02	7.84E-02	5.30E-03	0	0
3	500000	4.00E+02	1.58E+02	9.56E-03	8.49E-03	3.93E-57	1.78E-56
4	100000	0	0	0	0	0	0
5	500000	-12569.38	4.31E-02	-11033.14	3.58E+02	-12569.49	9.16E-12
6	500000	1.83E-02	8.44E-03	3.57E-02	5.06E-03	0	0
7	500000	5.41E-02	1.13E-02	9.19E-03	5.55E-04	5.35E-15	1.69E-15
8	500000	9.07E-02	3.30E-02	1.79E-02	2.30E-02	0	0
9	500000	4.94E-04	5.69E-04	1.42E-06	1.66E-07	1.57E-32	2.75E-47
10	500000	3.89E-03	4.09E-03	2.04E-05	2.92E-06	1.35E-32	2.48E-47
11	400000	6.76E-04	1.57E-04	5.27E-04	3.93E-04	4.04E-04	3.04E-04

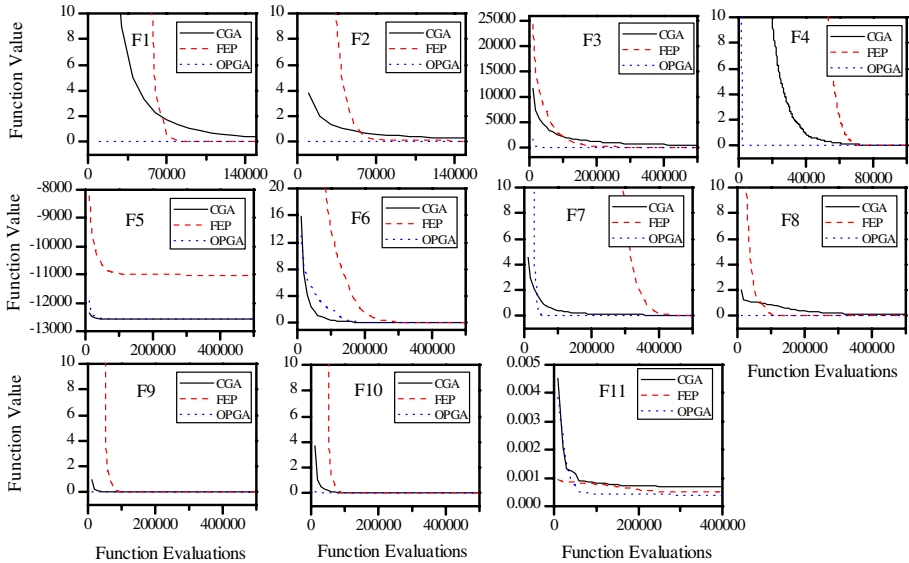


Fig. 4. Comparison among CGA, FEP, and OPGA on functions F1 to F11. All results were averaged over 100 runs.

Table 4 tabulates the means and standard deviations of the 100 independent runs to the test functions by CGA, FEP, and OPGA. Using the same maximum function evaluations (maxE) as termination conditions, the proposed OPGA outperforms CGA and FEP by achieving much better solutions and smaller standard deviations. Fig. 4 shows the convergence curves among CGA, FEP, and OPGA on the functions. The results present that OPGA can always find better results quicker than CGA and FEP.

5 Conclusion

This paper proposed a novel orthogonal predictive genetic algorithm (OPGA), which utilizes the orthogonal design method to predict the most promising direction for the best individual in GA to explore better results. The performance of the algorithm has been compared with the conventional genetic algorithm (CGA) and the fast evolutionary programming (FEP). The proposed OPGA outperforms the above two algorithms in both accuracy and convergence speed.

Acknowledgments. This work was supported by the National Science Foundation (NSF) of China (60573066) and the NSF of Guangdong (5003346), and SRF for ROCS, SEM, China.

References

1. Holland, J.H.: *Adaptation in Natural and Artificial Systems*, 2nd edn. MIT Press, Cambridge (1992)
2. Wang, W.Y., Li, Y.H.: Evolutionary Learning of BMF Fuzzy-neural Networks Using a Reduced-form Genetic Algorithm. *IEEE Trans. Syst. Man Cybern. Part B Cybern.* 33, 966–976 (2003)
3. Miller, J.A., Potter, W.D., Gandham, R.V., Lapena, C.N.: An Evaluation of Local Improvement Operators for Genetic Algorithms. *IEEE Trans. Syst. Man Cybern.* 23, 1340–1351 (1993)
4. Ishibuchi, H., Yoshida, T., Murata, T.: Balance Between Genetic Search and Local Search in Memetic Algorithms for Multiobjective Permutation Flowshop Scheduling. *IEEE Trans. Evol. Comput.* 7, 204–223 (2003)
5. Burke, E.K., Smith, A.J.: Hybrid Evolutionary Techniques for the Maintenance Scheduling Problem. *IEEE Trans. Power Syst.* 15, 122–128 (2000)
6. Denqiz, B., Altıparmak, F., Smith, A.E.: Local Search Genetic Algorithm for Optimal Design of Reliable Networks. *IEEE Trans. Evol. Comput.* 1, 179–188 (1997)
7. Baraglia, R., Hidalgo, J.I., Perego, R.: A Hybrid Heuristic for the Traveling Salesman Problem. *IEEE Trans. Evol. Comput.* 5, 613–622 (2001)
8. Tu, Z.G., Lu, Y.: A Robust Stochastic Genetic Algorithm (StGA) for Global Numerical Optimization. *IEEE Trans. Evol. Comput.* 8, 456–470 (2004)
9. Yao, X., Liu, Y.: Evolutionary Programming Made Faster. *IEEE Trans. Evol. Comput.* 3, 82–102 (1999)
10. Hedayat, A.S., Sloane, N.J.A., Stufken, J.: *Orthogonal Arrays: Theory and Applications*. Springer, New York (1999)

11. Montgomery, D.C.: Design and Analysis of Experiments, 5th edn. Wiley, New York (2000)
12. Hu, X.M., Zhang, J., Zhong, J.H.: An Enhanced Genetic Algorithm with Orthogonal Design. In: 2006 IEEE Congress on Evolutionary Computation, pp. 3174–3181. IEEE Press, New York (2006)

Appendix: Construction of the Orthogonal Array [10][11]

Step 1: Given an odd number s . Determine the smallest integer value of J satisfying

$$k = \frac{s^J - 1}{s - 1} \geq n \quad (\text{A.1})$$

Step 2: We have $N = s^J$. Construct the basic columns as follows

$$A_{ij} = \left\lfloor \frac{i - 1}{s^{J-p}} \right\rfloor \bmod s \quad (\text{A.2})$$

where $i = 1, 2, \dots, N$, $j = \frac{s^{p-1} - 1}{s - 1} + 1$, and $p = 1, 2, \dots, J$.

Step 3: Construct the non-basic columns as follows

$$\bar{A}_{j+(s-1)(t-1)+u} = (u\bar{A}_t + \bar{A}_j) \bmod s \quad (\text{A.3})$$

where $j = \frac{s^{p-1} - 1}{s - 1} + 1$, $p = 2, 3, \dots, J$, $t = 1, 2, \dots, j - 1$, $u = 1, 2, \dots, s - 1$.

Step 4: Increase A_{ij} by one

$$A_{ij} \leftarrow A_{ij} + 1 \quad (\text{A.4})$$

where $i = 1, 2, \dots, N$, $j = 1, 2, \dots, k$.

Phylogeny Inference Using a Multi-objective Evolutionary Algorithm with Indirect Representation

Md. Rafiul Hassan¹, M. Maruf Hossain¹, C.K. Karmakar²,
and Michael Kirley¹

¹ Department of Computer Science and Software Engineering

² Department of Electrical & Electronic Engineering

The University of Melbourne, Australia

{mrhassan,hossain,ckkarmakar,mkirley}@unimelb.edu.au

Abstract. The inference of phylogenetic trees is one of the most important tasks in computational biology. In this paper, we propose an extension to multi-objective evolutionary algorithms to address this problem. Here, we adopt an enhanced indirect encoding for a tree using the corresponding Prüfer code represented in Newick format. The algorithm generates a range of non-dominated trees given alternative fitness measures such as statistical likelihood and maximum parsimony. A key feature of this approach is the preservation of the evolutionary hierarchy between species. Preliminary experimental results indicate that our model is capable of generating a set of optimized phylogenetic trees for given species data and the results are comparable with other techniques.

1 Introduction

Phylogenetic inference is the construction of trees that represent the genealogical (evolutionary) relationships between different species. This task is a difficult computational problem (NP-complete) as the number of possible solutions (unrooted, bifurcating tree topologies) increases with the number of taxa included [1, 2]. Metaheuristic techniques, including evolutionary algorithms, offer a robust approach for tackling the phylogeny inference problem [3, 4, 5, 6]. However, an important consideration when using evolutionary algorithms is to adopt the most effective representation scheme to facilitate the search trajectory.

One of two alternative approaches are typically used to represent phylogenetic trees. In the first approach, a direct representation is used where the evolutionary algorithm conducts a search in the space of all possible phylogenetic trees. A well-known technique used to build the initial phylogenetic trees is the BIONJ algorithm [7]. In the second approach, an indirect method is used to encode the phylogenetic trees. For example, representations based on Prüfer code has been used with varying levels of success [8, 6, 9]. Here, the evolutionary operators manipulate this genotypic representation, which in turn is transformed into the phenotypic tree structures.

Recently, a number of papers have appeared that have used multi-objective algorithms for the phylogenetic inference problem (see Handl *et al.* [10] for an overview). Cancino and Delbem [4] have reported that a multi-objective approach was comparable with, and in some instances, outperformed single objective approaches. In this study, we continue this line of research. Here, we present a modified multi-objective evolutionary algorithm – PhyMOEA – using an enhanced indirect representation for the phylogeny inference problem. The underlying algorithm is an implementation of the NSGA-II algorithm [11], which operates on the phylogenetic trees (individuals) represented using Prüfer code and Newick string formatting. PhyMOEA generates a number of candidate trees, which are then evaluated using the non-dominated criteria in terms of maximum parsimony and likelihood. A key feature of our technique is the preservation of the evolutionary hierarchy between species.

The remainder of the paper is organized as follows: In the next section, we briefly review related work in phylogenetic inference. In Section 3, we describe the proposed model in detail. In Section 4, we present detailed experimental results. The paper concludes with a summary of the implications of this work.

2 Background

2.1 Phylogenetic Inference

The topology (or hierarchy) of the evolution of a set of given species (or taxa) can be represented by a phylogenetic tree. The construction of the tree is based on the processing of a data set consisting of characters representing each species. These characters might be amino acids or protein sequences, nucleotide (A, C, G and T) sequences or behavioural traits. For a number of taxa, the most common ancestor is considered as the root and the leaves are labeled by the taxa (see Fig 1 and 2). The evolution is represented by the branching of a parent node to the child node. Such trees are referred to as a “rooted tree.” Phylogenetic trees, in contrast, are referred to as “unrooted” trees [3]. That is, the directions of evolutionary changes correspond to a hypothetical ancestor rather than considering an explicit root.

Tree Representation Using Newick Format. Different string formats can be used to represent a phylogenetic tree. Perhaps the most widely used format is the Newick format. In this format, the hierarchy of taxa in the tree is represented by braces. Consider the example in Fig 3. The string depicted in Newick format illustrates the fact that taxa T1 and taxa T6 are at the same level and very close to each other. Taxa T3 is in a upper level of the tree, which connects with the level where T1 and T6 are connected. Thus, the Newick format for the whole tree can be used to represent existing hierarchies between given taxa.

Measuring Tree Quality. A number of metrics are available to evaluate the quality of a generated phylogenetic tree. Representative examples include the maximum parsimony and maximum likelihood criteria [1].

In the maximum parsimony approach, the parsimony score is calculated based on the minimum number of substitutions for each informative species. Consider

Taxa	Sequence										
	1	2	3	4	5	6	7	8	9	10	11
T1	A	A	G	A	G	T	G	C	A	C	A
T2	A	G	C	C	G	T	G	C	A	C	G
T3	A	G	C	G	G	C	C	T	G	C	A
T4	A	C	G	A	G	T	T	C	A	G	A
T5	A	A	C	A	C	T	G	C	C	C	G
T6	A	G	G	A	G	C	G	C	A	G	A

Fig. 1. An example of nucleotide sequences for 6 species. The degree of similarity of the sequences implies a particular genealogical relationship between the species, which in turn can be used to build the phylogenetic tree.

a small example: a dataset consisting of six taxa, each of which is represented by a sequence of 11 symbols as shown in Fig. 1. We have two trees for the above-mentioned six taxa (see Fig. 2). We can calculate which of the trees is better using the corresponding parsimony scores. For Tree 1 in Fig. 2(a), we obtain that the parsimony score of 24. This score is obtained by adding the numbers on each edge of the tree. The number on each of the edges are obtained by calculating the number of dissimilarities between the two sequences of the taxa connected by the edge. The parsimony score for Tree 2 in Fig. 2(b) is 25. Thus, these two scores signify that the Tree 1 is better “parsimoniously” than Tree 2.

In a maximum likelihood approach, the score is calculated using a statistical measure. The length (number) of each edge is calculated based on the conditional probability of a taxa given another connecting taxa. The likelihood score for the whole tree is thus calculated by multiplying all probabilities for each of the branches in the tree.

2.2 Related Work

Many of the studies focussed on the construction of phylogenetic trees have relied heavily on the BIONJ algorithm to build the initial trees (see e.g. [4, 12]).

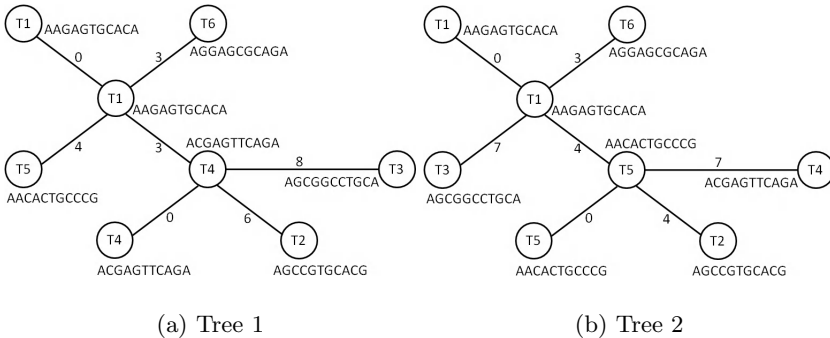


Fig. 2. Two possible phylogenetic trees for the dataset given in Fig. 1

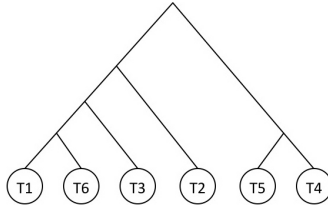


Fig. 3. Hierarchical representation of the tree $((((T1, T6), T3), T2), (T5, T4))$

Typically, these studies have concentrated on obtaining the “best trees.” For example, Poladian and Jermin [13] have attempted to optimise pre-built phylogenetic trees specifically in the case where different datasets provided conflicting information about the inferred best trees. In other work, the MRBAYES algorithm, which utilizes a Bayesian inference of phylogeny using a variant of a Markov chain, has been used to build the trees [14].

Saitou and Nei [15] proposed a neighbour-joining method for reconstructing phylogenetic trees based on evolutionary distance data. The principle underlying this method was to find pairs of Operational Taxonomic Units (data at the leaf node) that minimize the total branch length at each stage of the clustering of the units, starting with a star-like tree. Using this method, the branch lengths, as well as the topology, of a parsimonious tree can be obtained efficiently.

In related work, Takezaki and Nei [16] investigate the probability of obtaining the correct topology ($P(C)$) for these new distances, as well as for the traditional distance measures, by using computer simulation. They have used both the Infinite-Allele Model (IAM) and Stepwise Mutation Model (SMM), which are appropriate for determining the classical markers and microsatellite loci, respectively.

The first paper that employed a representation based on Prüfer code and evolutionary algorithms was reported by [8]. However, that model had significant shortcomings regarding the phylogenetic tree hierarchy. Related work where Prüfer coding has been used to build a spanning tree include studies by Gottlieb *et al.* [6] and [9]. Gottlieb and co-workers suggest that Prüfer code representation may in fact limit the effectiveness of the search in some instances. Cotta and Moscato [5] provide a comparative study of direct and indirect representation. They have suggested that the choice of representation (and corresponding operators) has a significant impact on the quality of solutions obtained. These studies reinforce the notion that any indirect representation used in an evolutionary algorithm must resolve the hierarchy issue between taxa if it is to produce (near-) optimal trees.

A limited number of papers have appeared in the literature that have employed multi-objective evolutionary algorithms for the phylogenetic inference problem. Handl *et al.* [10] provide an extensive survey of multi-objective optimisation in computational biology, with a limited discussion of phylogenetic trees applications. Recent examples from this domain include the work of Poladian and

Jermini [13], who optimised pre-built phylogenetic trees, specifically in the situation where different datasets provide conflicting information about the inferred “best trees”. The recent model of Cancino and Delbem [4] forms the starting point for the investigation reported in this paper (see details in the following sections).

3 The Model – PhyMOEA

3.1 Representation

Chromosome. An integer-based variable length chromosome is used to represent the tree using the corresponding Prüfer code. The length of the chromosome for each individual was set to $n - 2$, where n is the total number of taxa in a data file. Once the initial population was randomly generated, each of the chromosomes was translated to the corresponding phylogenetic tree. The challenge here was to generate an appropriate tree (represented in Newick format) so that the a “standard” tree optimisation program could be used.

Decoding the Prüfer Code. Several features of Prüfer coding suggests that this approach might support an efficient evolutionary search of the spaces in phylogenetic trees: the numbers can be decoded quickly; it is easy to generate random Prüfer code by choosing $n-2$ times from n node labels; and Prüfer coding supports conventional evolutionary operators, such as the k -point crossover and position-by-position mutation.

Consider the following example. Let $a_1 a_2 \dots a_{n-2} \in 1, \dots, n^{n-2}$ be a Prüfer code. Fig 4 provides a step-by-step example of tree generation from the corresponding Prüfer code. In this example, the Prüfer code is chosen as the sequence of 2, 4, 3, 3. Thus the Prüfer bar is obtained as [1, 5, 6, 2, 4, 3]. The Prüfer bar is obtained following the study of Cancino and Delbem [4]. For this example, the highest number would be 6, as the length of the Prüfer code is $n - 2 = 4$. We find the missing number from 1 to n (here, $n = 6$) in the Prüfer code. Thus, the contents of the Prüfer bar would be [1, 5, 6]. As the length of the Prüfer bar would be equal to n , for the remainder of the three cells in the Prüfer bar we scan the Prüfer code again. Initially, we get the number 2, and there is no repeat of this number in the remaining cells of the Prüfer code. Thus this number is added to the end of the Prüfer bar. Similarly, 4 is added to the Prüfer bar. Next we find 3, which has a repetition in the Prüfer code. We move forward to select the number in the next cell in the Prüfer code. Again, 3 is found, and there is no repetition of 3 in the remaining cells of the Prüfer code. Hence, 3 is added at the end of Prüfer bar. Finally, the Prüfer bar is found to be [1, 5, 6, 2, 4, 3].

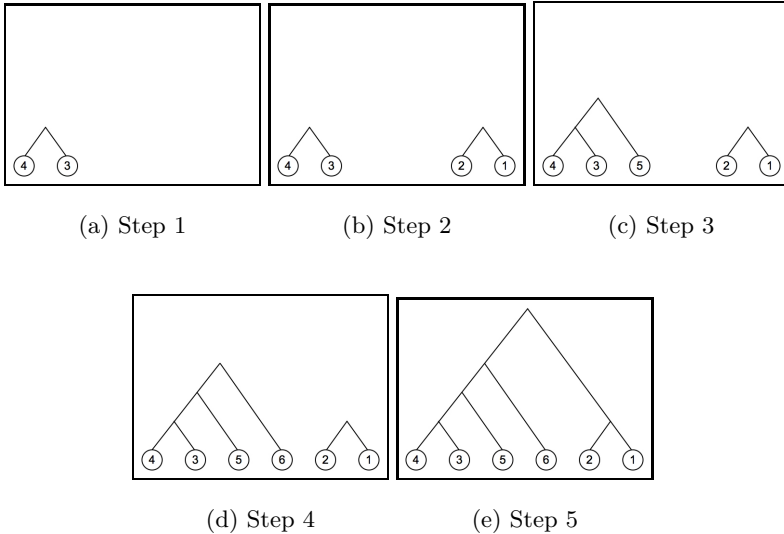
Now, we build the phylogenetic tree using the obtained Prüfer code and Prüfer bar. We take the two numbers from the last two cells of the Prüfer bar. We generate a tree by connecting these two numbers, as shown in Fig. 4. For the subsequent steps in building the tree, we consider each of the pairs of numbers, of which one is taken from the Prüfer code and the other from the Prüfer bar, such that each pair corresponds with the same cell location, starting from the first

Algorithm 1. PhyMOEA**Input(s):** *Prufer*: array of the Prüfer number, *PruferBar*: array of the Prüfer bar**Output:** *T*: Phylogenetic tree

```

1: Take the last two taxa from the Prüfer bar. Add taxa to tree.
2: for  $i \leftarrow 1$  to  $N - 2$  do
3:    $taxa_1 \leftarrow Prufer_i$ 
4:    $taxa_2 \leftarrow PruferBar_i$ 
5:   search for  $taxa_1$  and  $taxa_2$  in the tree
6:   if both taxa exists in tree then
7:     go to next iteration.
8:   end if
9:   if both taxa do not exist in tree then
10:    connect taxa root vertices and generate isolated phylogenetic tree.
11:   end if
12:   if  $taxa_1$  exists in tree then
13:     find the root of the subtree for  $taxa_1$ , then add  $taxa_2$  to the right of tree.
14:   end if
15:   if  $taxa_2$  exists in tree then
16:     find the root of the subtree for  $taxa_2$ , then add  $taxa_1$  to the right of tree.
17:   end if
18: end for
19: return T
20: end

```

**Fig. 4.** An example of generating the phylogenetic tree from a Prüfer code

cell up to $n - 2$ ($n - 2$ being the length of the Prüfer code). If any of the contents of a pair are found in the existing built tree, then the content is added to the tree. If the contents of a pair are not found in the existing built tree, we build an isolated tree by connecting the respective contents. Algorithm 1 provides the pseudocode to transform Prüfer code into corresponding phylogenetic tree.

3.2 Genetic Operators

Two genetic operators are used in the PhyMOEA. The crossover operator is a version of standard two-point crossover for integer-based string representation. The mutation operator is based on the well-known Nearest Neighbour Interchange (NNI) [17], alongside the other topological modification algorithms used in phylogenetic inference. We have adopted this approach as NNI is widely used due to the fact that it performs minimal tree modifications. This operator then carries out the following steps [4]:

- Select a branch connecting the nodes i, j that defines two pairs of neighbours: w, x adjacent to $i(w, x \neq j)$ and y, z adjacent to $j(y, z \neq i)$.
- Swap the nodes between each pair of neighbours.

To improve the phylogenetic tree likelihood value, branch lengths are optimized. To do this, randomly selected branches have their lengths modified. Here, the branch length is multiplied by a factor obtained from a gamma distribution [18].

3.3 Fitness Evaluation and Selection

The parsimony and likelihood scores are calculated using Fitch [19] and Felsenstein [20] algorithms, respectively. Appropriate crowding and non-dominated sorting features of NSGA-II are then used during the selection phase.

4 Experiments and Results

To evaluate the performance of PhyMOEA, we have used three nucleotide datasets from [4]. The RDPII_218 dataset is comprised of 218 prokaryotic RNA sequences (4182 sites) taken from the Ribosomal Database Project II [21]. The rbcL_55 dataset has sequences of the rbcL chloroplast gene from 55 species of green plants (1314 sites) [18]. Finally, the mtDNA_186 dataset contains 186 human mitochondrial DNA sequences (16608 sites) taken from the Human Mitochondrial Genome Database (mtDB) [22].

The maximum parsimony and likelihood analyses were performed using the DNAPars [23] and DNAML [24] programs, respectively from the PHYLIP Package [2]. These programs are widely used and have been shown to be reliable. Table 1 lists the parameters used in the evolutionary algorithm. Table 2 shows the maximum parsimony and likelihood results obtained from these programs for all the datasets considered. Table 3 shows the performance improvement of PhyMOEA compared with PHYLIP as used by Cancino and Delbem [4]. However, where Cancino and Delbem generated 1000 initial trees and then used tools from PHYLIP package (DNAPars and PAML) to optimise and obtain best trees, we have only used the package to obtain scores of our generated trees.

Table 1. Multi-objective evolutionary algorithm parameters

Parameter	Values
Generations	100
Population size	48
Crossover rate	0.9
Mutation rate	0.033
Evolution Model	HKY85
Chromosome length	RDPII_218 – 216, rbcL_55 – 53, mtDNA_186 – 184

Table 2. Results for the PhyMOEA

Dataset	# of best tree on Pareto-front	# of final trees	Best parsimony score		Best likelihood score	
			likelihood	parsimony	likelihood	parsimony
RDPII_218	5	15	−157870.7	40728	−156314.7	40720
rbcL_55	3	21	−24577.2	4866	−24576.06	4871
mtDNA_186	2	32	−40389.66	2430	−40387.01	2432

Table 3. Results Comparison

Dataset	PhyMOEA		PHYLP			
			Best parsimony score (DNAPars)		Best likelihood score (DNAML)	
	Best likelihood	Best parsimony	likelihood	parsimony	likelihood	parsimony
RDPII_218	−156314.7	40728	−157924.1	42736	−157929.06	42756
rbcL_55	−24576.06	4866	−24657	4881	−24588.87	4888
mtDNA_186	−40387.01	2430	−41392.04	2467	−41391.06	2471

5 Concluding Comments

In this paper, we have proposed and evaluated a modified technique for generating phylogenetic trees using a multi-objective evolutionary algorithm. The contribution of our study is two-fold. Firstly, a novel approach was proposed for generating the initial phylogenetic topology that did not rely solely on the string representation required when using the BIONJ algorithm. Secondly, we have demonstrated that our multi-objective evolutionary algorithm can be used to find a set of trade-off phylogenetic trees for given species data using a small number of iterations and thereby reducing the execution time.

PhyMOEA resolves one of the existing difficulties – to represent the initial tree in numeric format – when using evolutionary algorithms to solve the phylogenetic inference problem. The representation used in PhyMOEA is based on Prüfer code, which are transformed to phylogenetic tree topologies with hierarchies. The results of the two objectives considered – the maximum parsimony and likelihood scores – are impressive for the datasets. For example, on rbcL_55 dataset PhyMOEA performed better when compared to PHYLP (see Table 3)

and PhyloMOEA [4]. PhyMOEA improves both parsimony and likelihood scores, where as existing techniques focused on obtaining “best trees” in terms of either parsimony or likelihood scores. Similar trends in the result were also found for the other datasets.

In our model, when generating the initial tree topology, we did not consider the actual dataset and hence a diverse set of randomly generated phylogenetic trees were generated. Such diversity adds flexibility, promotes search space exploration and thus contributes hopefully to the generation of better solutions. Clearly, this approach has merit and the preliminary results reported in this study suggest that the proposed Prüfer code based multi-objective evolutionary method for phylogenetic inference warrants further investigation.

References

1. Baldauf, S.L.: Phylogeny for the faint of heart: A tutorial. *Trends in Genetics* 19(6), 345–351 (2003)
2. Felsenstein, J.: PHYLIP – Phylogeny Inference Package (2000), <http://evolution.genetics.washington.edu/phylip.html>
3. Congdon, C.B., Septor, K.J.: Phylogenetic trees Using Evolutionary Search: Initial Progress in Extending Gaphyl to Work with Genetic Data. In: *The 2003 Congress on Evolutionary Computation (CEC 2003)*, vol. 1, pp. 320–326 (December 2003)
4. Cancino, W., Delbem, A.C.B.: A Multi-Objective Evolutionary Approach for Phylogenetic Inference. In: Obayashi, S., Deb, K., Poloni, C., Hiroyasu, T., Murata, T. (eds.) *EMO 2007. LNCS*, vol. 4403, pp. 428–442. Springer, Heidelberg (2007)
5. Cotta, C., Moscato, P.: Inferring phylogenetic trees using evolutionary algorithms. In: Guervós, J.J.M., Adamidis, P.A., Beyer, H.-G., Fernández-Villacañas, J.-L., Schwefel, H.-P. (eds.) *PPSN 2002. LNCS*, vol. 2439, pp. 720–729. Springer, Heidelberg (2002)
6. Gottlieb, J., Julstrom, B.A., Raidl, G.R., Rothlauf, F.: Prüfer Numbers: A Poor Representation of Spanning Trees for Evolutionary Search. In: Spector, L., Goodman, E.D., Wu, A., Langdon, W.B., Voigt, H., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M.H., Burke, E. (eds.) *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, pp. 343–350. Morgan Kaufmann, San Francisco (2001)
7. Gascuel, O.: BIONJ: an improved version of the NJ algorithm based on a simple model of sequence data. *Molecular Biology and Evolution* 14(7), 685–695 (1997)
8. Reijmers, T.H., Wehrens, R., Buydens, L.M.C.: Quality Criteria of Genetic Algorithms for Construction of Phylogenetic Trees. *Journal of Computational Chemistry* 20(8), 867–876 (1999)
9. Gen, M., Li, Y.: Spanning tree-based genetic algorithm for the bicriteria fixedcharge transportation problem. In: *Proceedings of the 1999 Congress on Evolutionary Computation (CEC 1999)*, Washington, DC, USA, vol. 3, p. 2271 (1999)
10. Handl, J., Kell, D.B., Knowles, J.: Multiobjective Optimization in Bioinformatics and Computational Biology. *IEEE/ACM Transaction on Computational Biology and Bioinformatics* 4(2), 279–292 (2007)
11. Deb, K., Agarwal, S., Pratab, A., Meyarivan, T.: A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. KanGAL report 2000001, Indian Institute of Technology, Kanpur, India (2000)

12. Zelwer, M., Daubin, V.: Detecting phylogenetic incongruence using BIONJ: an improvement of the ILD test. *Molecular Phylogenetics and Evolution* 33(3), 687–693 (2004)
13. Poladian, L., Jermini, L.S.: Multi-objective evolutionary algorithms and phylogenetic inference with multiple data sets. *Soft Computing* 10, 359–368 (2006)
14. Huelsenbeck, J.P., Ronquist, F.: MRBAYES: Bayesian inference of phylogenetic trees. *Bioinformatics* 17(8), 754–755 (2001)
15. Saitou, N., Nei, M.: The Neighbor-Joining Method: A New Method for reconstructing Phylogenetic Trees. *Molecular Biology and Evolution* 4(4), 406–425 (1987)
16. Takezaki, N., Nei, M.: Genetic distances and reconstruction of phylogenetic trees from microsatellite DNA. *Genetics* 144(1), 389–399 (1996)
17. Swofford, D., Olsen, G., Waddell, P., Hillis, D.: Phylogeny Reconstruction. In: *Molecular Systematics*, 3rd edn., pp. 407–514. Sinauer (1996)
18. Lewis, P.O.: A Genetic Algorithm for Maximum-Likelihood Phylogeny Inference Using Nucleotide Data. *Molecular Biology and Evolution* 15(3), 277–283 (1998)
19. Fitch, W.: Toward Defining the Course of Evolution: Minimum Change for a Specific Tree Topology. *Systematic Zoology* 20(4), 406–416 (1972)
20. Felsenstein, J.: Evolutionary Trees from DNA Sequences: A Maximum Likelihood Approach. *Journal of Molecular Evolution* 17, 368–376 (1981)
21. Cole, J., Chai, B., Farris, R., Wang, K.S., McGarrell, D., Garrity, G., Tiedje, J.: The Ribosomal Database Project (RDP-II): Sequences and Tools for High-throughput rRNA Analysis. *Nucleic Acids Research* 33, D294–D296 (2005)
22. Ingman, M., Gyllenstein, U.: mtDB: Human Mitochondrial Genome Database, a Resource for Population Genetics and Medical Sciences. *Nucleic Acids Research* 34, D749–D751 (2006)
23. Felsenstein, J.: DNAPARS – DNA Parsimony Program (1996),
<http://cmgm.stanford.edu/phylip/dnapars.html>
24. Felsenstein, J.: DNAML – DNA Maximum Likelihood program (1993),
<http://cmgm.stanford.edu/phylip/dnaml.html>

Evolved Look-Up Tables for Simulated DNA Controlled Robots

Gary Greenfield

Mathematics & Computer Science
University of Richmond
Richmond VA 23173, USA
ggreenfi@richmond.edu

<http://www.mathcs.richmond.edu/~ggreenfi/>

Abstract. We describe our efforts to convert (short) DNA sequences obtained from the NCBI library into control sequences for simulated robots by simultaneously evolving both a look up table to assign codons to robot commands and a look up table to assign codons to numerical values that serve as arguments to those commands. Our simulated robot is loosely modeled after the Khepera robot. When the robot's sensing capabilities are disabled, we are provided with a sophisticated turtle graphics platform. We formulate a fitness function for evaluating the drawings obtained from codon look up tables and we make inter-DNA and intra-DNA comparisons using our evolved tables. Our results suggest that information content can only be weakly extracted from DNA in this way.

1 Introduction

Recently, Latham et al. [1] used DNA sequences as “biological software” to generate 3D forms using their well known FormGrow system [2]. To do so they mapped codons to shape grammar operands (e.g. Grow, Bend, Twist, etc. functions) and also mapped codons to the numeric arguments needed for these commands. When describing the table that transforms codons to operands they wrote, “We experimented with producing a few different iterations of the transform table in order to to get a balance of functions that would produce a visually interesting variety of shapes.” [1, p. 437]. Their codon mapping is coarse in the sense that only the first two nucleotides of a codon are used to determine which of the seven available shape grammar operands will be assigned. To map codons to numerical arguments they simply list all 64 codons in lexicographic order and assign to them the integers 0 through 63 viz. $AAA \rightarrow 0$, $AAC \rightarrow 1$, $\dots$, $TTT \rightarrow 63$. Their results are striking, and their application — using morphing to compare the shapes generated from reconstructed ancestral sequences of the protein Argininosuccinate lyase — ingenious.

In this paper we address the question of how to make the Latham technique of realizing DNA sequences as biological software more general purpose. Thus

we use the genetic algorithm to evolve the look up tables needed for converting codons to operands and arguments. Following Alife principles, we use a less sophisticated rendering scheme as our imaging platform. The visualizations realized from our codon mappings are simple line drawings made by a simulated robot. In this respect our drawings are closer in spirit to the drawings of the physical robots made by Bird et al. [3]. Our robot simulation platform is a simplified version of the one we used in [4] to obtain “robot paintings” using evolutionary methods such as the one shown in Figure 1. We remark that robot paintings such as the one in Figure 1 are a far cry from the state of the art as exemplified by Moura, Ramos, and Pereira (see [5] [6] [7]).

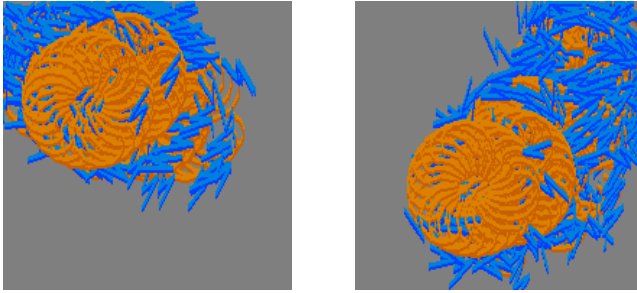


Fig. 1. Evolved robot paintings (see [4])

This paper is organized as follows. In section two we provide some background on the use of evolutionary techniques for aesthetic purposes. In section three we give the specifications for our simulated robot. In section four we describe our controller. In section five we present the artificial genetics we use to evolve look up tables. In section six we outline our evolutionary framework. In section seven we develop our fitness function. In section eight we show some of the drawings we obtained by evolving look up tables. In section nine we give some inter and intra organism comparison examples. In section ten we offer our summary and conclusions.

2 Background

The use of the (non-interactive) genetic algorithm in evolutionary art was first considered by Baluja et al. [8]. By using this technique for evolving two-dimensional imagery, interesting results have been obtained by Greenfield using co-evolution [9] coupled with the image generation method known as “evolving expressions” [10], by Machado and Cardosa [11] using neural nets, and by Bentley [12] in trying to identify cellular automata “patterns.” In general, the question of how to evaluate aesthetics on the basis of scientific principles and computational methodologies is a difficult one. The tenet we shall follow here is that rudimentary geometric measurements can be used to help identify image organization and complexity.

3 Simulated Robot Specification

The design of our simulated robot is loosely based on a software model for Khepera robot simulation by Harlan et al. [13]. Thus our robot is a (virtual) circular drawing agent whose forward direction is determined by the *unit* vector (d_x, d_y) . It has two pens that operate asynchronously so that either none, one of the two, or both may be down at any given instant. A robot's mark can be up to five units wide when both pens are down. However, during simulated evolution for fitness evaluation purposes we will restrict the mark to being only one unit wide. This will be explained further below. The robot can swivel (i.e rotate in place) no more than 10° clockwise or counterclockwise per clock cycle and can move s units per clock cycle, $-1 \leq s \leq 1$, in either the forward or backward direction in accordance with the sign of s . Since we are interested only in drawing capabilities, not sensing capabilities, the four binary valued proximity sensors and the three-channel color sensor are disabled. The robot roams on an $n \times n$ grid. Here we will always use $n = 400$.

4 The Simulated Robot Controller

Our robot is managed by an interrupt driven controller whose job is to place a sequence of commands in an execution queue, sleep until the queue is empty, and then plan and load the robot's next sequence of commands when it is awoken. Here we use a null controller that simply loads the commands afforded by the DNA sequence and halts the robot if the sequence is completed before the time limit expires. Our robot has five commands that can be executed/loaded:

MOV	Move
SWI	Swivel
SPD	Set Speed
PUP	Pen Up
PDN	Pen Down

Each of these commands requires a single numeric argument. Since codons used as arguments will furnish values in the range 0 to 63 some remapping is necessary as follows. If the codon look up table returns the numeric value v then the argument is $1 + v$ for MOV, $5(v - 32)$ for SWI, $0.03(v - 32)$ for SPD, and for the pen commands 0 if $v \in [0, 15]$ and 1 if $v \in [16, 63]$. Thus the range for move is 1 to 64, swivel -160 to 155 in increments of 5, and speed -0.96 to 0.93 in increments of 0.03. The reason that our MOV command is adjusted so that a move is always nonzero, is so that degenerate drawings that would foil our evaluation scheme will not emerge. For the same reason a request for a SPD of 0.0 is adjusted to a "creep" speed of 0.001. The point is, a robot will not be able to exhaust the DNA sequence by setting speed to zero and executing a host of move commands. Note that by calculating how many clock cycles various actions will take, discrete event simulation can be used to execute the robot drawing.

	A	C	G	T
A	MOV	MOV	PDN	PDN
G	PUP	PUP	SWI	PUP
C	MOV	SWI	SWI	SPD
T	SPD	PDN	MOV	SWI

	A	C	G	T	
A	0	4	8	12	A
C	1	5	9	13	C
G	2	6	10	14	G
T	3	7	11	15	T
A	16	20	24	28	A
C	17	21	25	29	C
G	:	:	:	:	
T	:	:	:	:	
A	32	36	40	44	A
C	33	37	41	45	C
G	34	38	42	46	G
T	35	39	43	47	T
A	48	52	56	60	A
C	49	53	57	61	C
G	50	54	58	62	G
T	51	55	59	63	T

Fig. 2. Example of a look-up table for commands (left) and arguments (right)

5 Artificial Genetics for Look Up Tables

Following [1], to represent the codon to operand mapping we use a 4×4 array whose rows (respectively columns) are labeled A, C, G, T and whose entries are numeric equivalents for the robot commands i.e. 0,1,2,3,4. Thus, recalling that the only first two nucleotides of a codon are needed, an evolved codon to operand look up table will take the form of the table on the left in Figure 2 where for ease of reading we have used the symbolic equivalents. One point crossover for two such tables is performed by writing the tables out as one dimensional arrays, performing one point crossover in the usual manner, and copying the results back into the two dimensional arrays. Point mutation is used to replace an entry with a new entry. The probability an entry is chosen for point mutation is set to 1/16. For use with the genetic algorithm, these tables are initialized with random integer values between 0 and 4 inclusive. For the codon to argument mapping, we initialize the table sequentially exactly the way Latham et al. did using the values 0 to 63, so the layout can be visualized either as a two dimensional array as shown above in the table on the right in Figure 2 or, with the row index fixed for each “slab”, as four arrays as follows:

0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60
1	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61
2	6	10	14	18	22	26	30	34	38	42	46	50	54	58	62
3	7	11	15	19	23	27	31	35	39	43	47	51	55	59	63

To implement crossover, we copy to and from the $3 \times 3 \times 3$ array to a one dimensional array in such a way that the sequential ordering of this initial assignment is preserved. For point mutation, the probability an entry is selected is set to 3/100.

6 Evolutionary Framework

After the DNA sequence is loaded for each genome the look up tables are loaded, the robot is placed in the center of the grid with a heading of 45° i.e. northeast,

and the robot is permitted to execute commands for 250,000 clock cycles. Our evolutionary framework uses a population of $P = 40$ genomes. For each of $G = 400$ generations, the drawing made using genome g is assigned fitness F_g using the calculation described below. Then the $P/2$ least fit genomes are discarded and $P/4$ breeding pairs are formed by cloning from the pool of $P/2$ survivors. Breeding within each pair is performed as described above. Thus an evolutionary run considers $P + G(P/2) = 40 + 400 \cdot 20 = 8040$ robot drawings. The drawing associated with the most fit genome is logged after every 100 generations. Note that this scheme is elitist. Perhaps because of this, preliminary experiments showed that fitness seldom increased after 400 generations. Moreover, by that point a significant number of matings produced robots with zero fitness. This suggests a local maximum in the fitness landscape has been located.

7 The Fitness Calculation

Let C_v be the number of squares of the grid the robot visited, C_p be the number of squares the robot drew on, and C_w the number of attempts by the robot to cross a grid boundary i.e. number of wall collisions. Genome g is assigned fitness F_g where

$$F_g = \begin{cases} 0 & \text{if } C_w > 1000 \\ \frac{-4C_p(C_p - C_v)}{C_v^2} C_v & \text{otherwise} \end{cases}$$

The condition on C_w is imposed in order to penalize the robot for repeatedly trying to wander off the grid. Otherwise, the objective is to maximize the number of squares visited while drawing approximately half the time. Note that the fraction multiplied by C_v is a normalization term that varies from 0 to 1 as C_p varies from 0 to C_v and it assumes a maximum value of 1 when $C_p = C_v/2$. For this fitness calculation we use a mark only one unit wide to be faithful to the

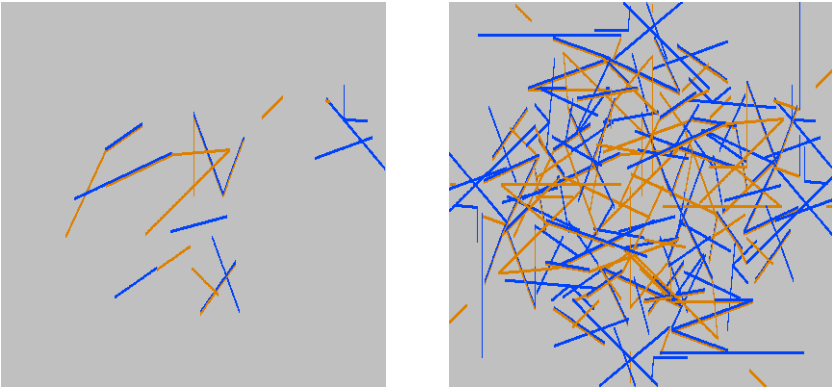


Fig. 3. Left: a motif evolved from a 3000 codon segment of *E. coli* DNA. Right: the motif iterated by using 9 copies of this segment to control the drawing robot.

parameter assessments. In more descriptive terms, we are seeking to evolve look up tables that furnish a visually interesting DNA programmed “motif” that lies wholly within the grid. As an example, the left image of Figure 3 shows the motif we evolved using a 3000 codon *E. coli* DNA segment, while the right image shows the type of art work we envision using such motifs. It was obtained by letting the robot draw this motif using a controller driven by 9 copies of this DNA segment. In that case, the robot executed commands for $(9)(250000) = 3250000$ clock cycles.

8 Results from Different DNA Sequences

We selected seven DNA sequences from The National Center for Biotechnology Information (NCBI) gene bank at <http://www.ncbi.nlm.nih.gov>. Their names, the codon lengths of the initial segments we used, and their NCBI identification numbers are shown in the table below. We also included a pseudorandomly generated sequence of 1000 codons as a control, 1000 being approximately the average length of the seven we chose from the gene bank.

Organism	Codons	Accession
<i>Alloctrotus fragilis</i> (urchin)	213	AY221005
<i>Anopheles gambiae</i> (mosquito)	575	EQ097731
<i>E. coli</i> (bacteria)	3000	NC009838
<i>Gallus gallus</i> (chicken)	323	AB434917
<i>Lentinula edodes</i> (shitake)	2785	DQ973527
<i>Mus musculus</i> (mouse)	679	EU597798
<i>Zea mays</i> (corn)	211	EU137662

For each organism we ran simulated evolution three times, thus obtaining three sets of tables corresponding to three robot drawings to choose from. We pseudocolored the paths the robots took to indicate when the robot was moving with both pens up (black), when the robot was in creep mode (green), and when the robot was drawing with at least one pen engaged in drawing (blue or orange). Figure 4 shows the three path drawings we obtained using the corn DNA.

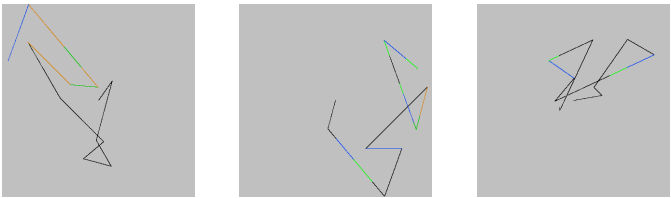


Fig. 4. Robot path drawings obtained from three separate evolutionary runs using corn DNA

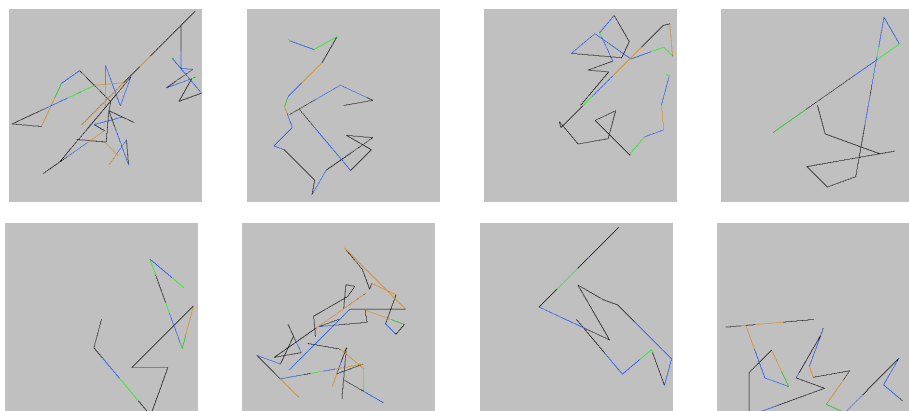


Fig. 5. Evolved DNA robot path drawings. Top row from left: *E. coli*, mouse, anopheles, chicken. Bottom row from left: corn, shitake, urchin, random.

Due to an oversight, for path drawings, we failed to distinguish between the cases where pen #1 was down and both pen #0 and pen #1 were down. Both cases yield a blue track. The reason for this oversight was because the criterion we used for choosing from among the three path drawings was the number of distinct contiguous marks the robot drew. Thus, even though the fitnesses (from left to right) of the path drawings in Figure 4 are 1426, 1244, and 1151 we selected the middle drawing. Figure 5 shows the path drawings selected on this basis that we obtained using the genetic algorithm for all eight of our DNA samples. Figure 6 shows the resulting DNA motifs.

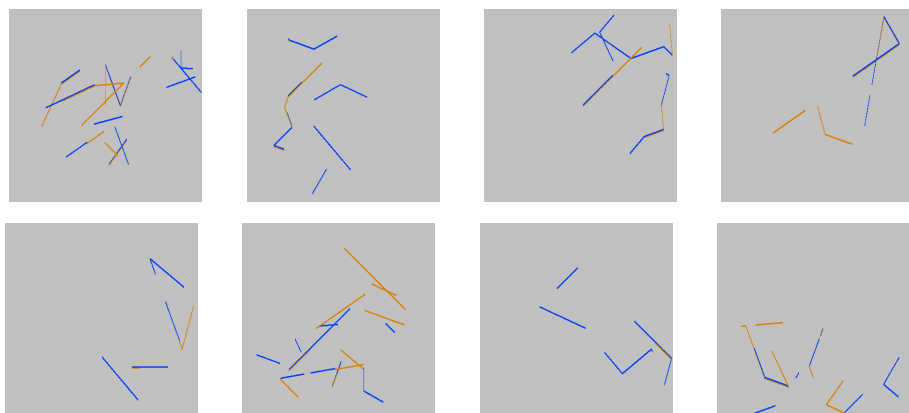


Fig. 6. The DNA motifs extracted from the previous figure's path drawings. The sequencing of the images is the same.

9 Some Comparison Results

Since our *E. coli* DNA sample has thousands upon thousands of codons, we were able to apply the tables we evolved from the initial segment of 3000 codons to eight successive segments of 3000 codons. Half of these either had zero fitness

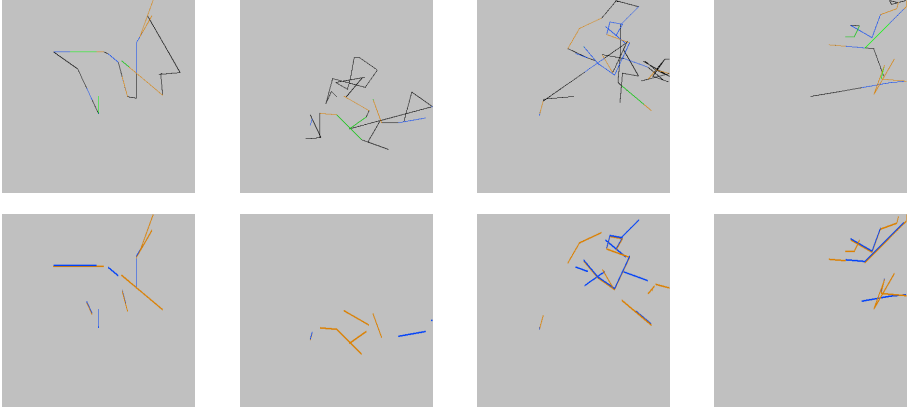


Fig. 7. Robot path drawings (top row) and their corresponding DNA motifs (bottom row) from 4 additional 3000 codon *E. coli* DNA segments that were rendered using the evolved tables obtained using the initial 3000 codon segment

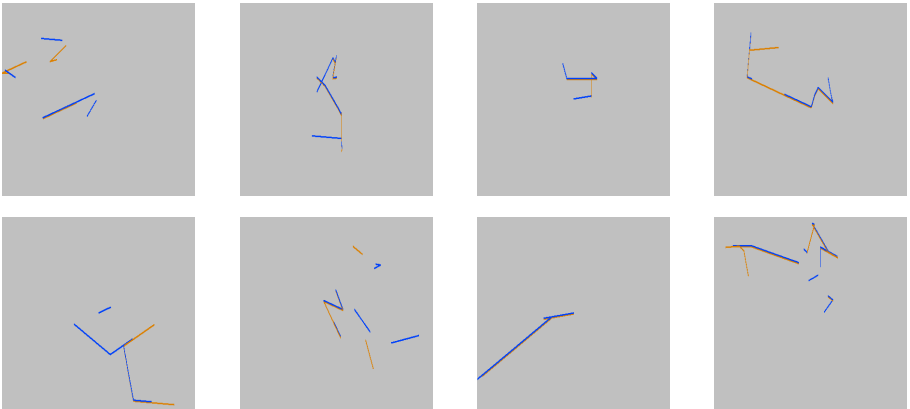


Fig. 8. Some inter organism DNA robot motifs. Top row from left: *E. coli* DNA with shitake tables, mouse DNA with anopheles tables, anopheles DNA with chicken tables, chicken DNA with anopheles tables. Bottom row from left: corn DNA with anopheles tables, shitake DNA with *E. coli* tables, urchin DNA with corn tables, random DNA with anopheles tables.

or produced very little robot movement. The four most interesting path drawings and their motifs are shown in Figure 7.

We also tried using each DNA sequence with all the other sets of evolved tables. Space prohibits showing all of the results, but we have collected in Figure 8 the example for each DNA sequence that we found most interesting. It is amusing that tables evolved from the pesky mosquito were the ones we chose most often.

10 Summary, Conclusions, and Future Work

We have described how we used evolutionary techniques to evolve look up tables for translating codon sequences of organismal DNA into drawing command sequences for a simulated Kherpera-like robot. We developed a method for evaluating the resulting drawings on the basis of aesthetics. We provided examples from a small test suite and investigated the utility of evolved tables via inter and intra organism comparisons.

The drawings we evolved are not very impressive. No doubt this is explained in large part by the limited potential of our underlying generative system. Our drawings are on par with those of Bird et al. [3]. Our visualizations can not compete with those of Latham et al. [1]. This raises a question that probably has no definitive answer: Are impressive visualizations that are DNA driven attributable more to the information content being provided by the DNA or to the expressive characteristics of the underlying generative system?

There are three natural opportunities for further research. First, our fitness function did not take into account our second-order aesthetic criterion of exhibiting a number of different contiguous paths in the drawing. One possibility for addressing this is to use multi-objective optimization. The only previous work we are aware of in the area had limited success with this approach [14]. One of the reasons is that it runs counter to the rationale for using evolutionary techniques in the first place — reducing the number of aesthetic decisions a human must make. Second, it would be of interest to formulate a statistical design for analyzing the utility of our evolved tables. Logic suggests that our drawing results must be extremely sensitive to key codon sequences occurring very early in the DNA samples we were using. A DNA sequence and a pair of look up tables will yield somewhere between 100 and 1500 drawing commands. It is nearly as challenging to analyze this sequence as the DNA itself. Third, the reader may have noticed that we did not take into account start and stop codons, junk DNA, introns, exons and a host of other issues surrounding how to read and interpret DNA. The experts we consulted gave confusing and often conflicting advice on such matters so in the end we elected to treat all codons as equally valuable. This may have been naive and future work could test this hypothesis.

References

1. Latham, W., Shaw, M., Todd, S., Leymarie, F., Jefferys, B., Kelly, L.: Using dna to generate 3d organic art forms. In: Giacobini, M., Brabazon, A., Cagnoni, S., Di Caro, G.A., Drechsler, R., Ekárt, A., Esparcia-Alcázar, A.I., Farooq, M., Fink, A., McCormack, J., O'Neill, M., Romero, J., Rothlauf, F., Squillero, G., Uyar, A.Ş., Yang, S. (eds.) *EvoWorkshops 2008*. LNCS, vol. 4974, pp. 433–442. Springer, Heidelberg (2008)
2. Todd, S., Latham, W.: *Evolutionary Art and Computers*. Academic Press, London (1992)
3. Bird, J., Husbands, P., Perris, M., Bigge, B., Brown, P.: Implicit fitness functions for evolving a drawing robot. In: Giacobini, M., Brabazon, A., Cagnoni, S., Di Caro, G.A., Drechsler, R., Ekárt, A., Esparcia-Alcázar, A.I., Farooq, M., Fink, A., McCormack, J., O'Neill, M., Romero, J., Rothlauf, F., Squillero, G., Uyar, A.Ş., Yang, S. (eds.) *EvoWorkshops 2008*. LNCS, vol. 4974, pp. 473–478. Springer, Heidelberg (2008)
4. Greenfield, G.: Robot paintings evolved using simulated robots. In: Rothlauf, F., Branke, J., Cagnoni, S., Costa, E., Cotta, C., Drechsler, R., Lutton, E., Machado, P., Moore, J.H., Romero, J., Smith, G.D., Squillero, G., Takagi, H. (eds.) *EvoWorkshops 2006*. LNCS, vol. 3907, pp. 611–621. Springer, Heidelberg (2006)
5. Moura, L., Pereira, H.: *Man + Robots: Symbiotic Art*. Institut d'Art Contemporain, Lyon/Villeurbanne, France (2004)
6. Moura, L., Ramos, V.: Swarm paintings — nonhuman art. In: Maubant, J. (ed.) *Architopia: Book, Art, Architecture, and Science*, Lyon/Villeurbanne, France, Institut d'Art Contemporain, pp. 5–24 (2002)
7. Ramos, V.: Self-organizing the abstract: canvas as a swarm habitat for collective memory, perception and cooperative distributed creativity. In: Rekalde, J. (ed.) *First Art & Science Symposium, Models to Know Reality*, Bilbao, Spain, p. 59 (2003)
8. Baluja, S., Pomerleau, D., Jochem, T.: Towards automated artificial evolution for computer-generated images. *Connection Science* (6), 325–354 (1994)
9. Greenfield, G.: On the co-evolution of evolving expressions. *International Journal of Computational Intelligence and Applications* 2(1), 17–31 (2002)
10. Sims, K.: Artificial evolution for computer graphics. *Computer Graphics* 25, 319–328 (1991)
11. Machado, P., Cardoso, A.: Computing aesthetics. In: de Oliveira, F.M. (ed.) *SBIA 1998*. LNCS (LNAI), vol. 1515, pp. 219–229. Springer, Heidelberg (1998)
12. Bentley, K.: Exploring aesthetic pattern formation. In: Soddu, C. (ed.) *Proceedings of the Fifth International Conference of Generative Art*. Alea Press, Milan (2002), <http://www.generativeart.com/papersGA2002/20.pdf>
13. Harlan, R., Levine, D., McClarigan, S.: The khepera robot and krobot class: a platform for introducing robotics in the undergraduate curriculum. In: *Technical Report 4*, St. Bonaventure Undergraduate Robotics Laboratory, St. Bonaventure, NY, St. Bonaventure University (2000)
14. Greenfield, G.: Evolving aesthetic images using multiobjective optimization. In: *Congress on Evolutionary Computation, CEC 2003*, Canberra, Australia, New York, NY, December 9–12, pp. 1903–1909. IEEE Press, Los Alamitos (2003)

Multi-objective Improvement of Software Using Co-evolution and Smart Seeding

Andrea Arcuri¹, David Robert White², John Clark², and Xin Yao¹

¹ The Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), The School of Computer Science, The University of Birmingham, Edgbaston, Birmingham, B15 2TT, UK

² Department of Computer Science, University of York, YO10 5DD, UK

Abstract. Optimising non-functional properties of software is an important part of the implementation process. One such property is execution time, and compilers target a reduction in execution time using a variety of optimisation techniques. Compiler optimisation is not always able to produce semantically equivalent alternatives that improve execution times, even if such alternatives are known to exist. Often, this is due to the local nature of such optimisations. In this paper we present a novel framework for optimising existing software using a hybrid of evolutionary optimisation techniques. Given as input the implementation of a program or function, we use Genetic Programming to evolve a new semantically equivalent version, optimised to reduce execution time subject to a given probability distribution of inputs. We employ a co-evolved population of test cases to encourage the preservation of the program's semantics, and exploit the original program through seeding of the population in order to focus the search. We carry out experiments to identify the important factors in maximising efficiency gains. Although in this work we have optimised execution time, other non-functional criteria could be optimised in a similar manner.

1 Introduction

Software developers must not only implement code that adheres to the customer's functional requirements, but they should also pay attention to performance details. There are many contexts in which the execution time is important, for example to aid performance in high-load server applications, or to maximise time spent in a power-saving mode in software for low-resource systems. Typical programmer mistakes may include the use of an inefficient algorithm or data structure, such as employing an $\Theta(n^2)$ sorting algorithm.

Even if the correct data structures and algorithms are employed, their actual implementations might still be improved. In general, compilers cannot restructure a program's implementation without restriction, even if employing semantics-preserving transformations. The alternative of relying on manual optimisation is not always possible: the performance implications of design decisions may be dependent on low-level details hidden from the programmer, or be subject to subtle interactions with other properties of the software.

To complicate the problem, external factors contribute to the execution time of software, such as operating system and memory caches events. Taking into account these

factors is difficult, and so compilers usually focus on optimising localised areas of code, rather than restructuring entire functions.

More sophisticated optimisations can be applied if we take into account the probability distribution of the *usage* of the software. For example, if a function takes an integer input and if we know that this input will usually be positive, this information could be exploited by optimising the software for positive input values.

In this paper we present a novel framework based on evolutionary optimisation techniques for optimising software. Given the code of a function as input to the framework, the optimisations are performed at the program level and consider the probability distribution of inputs to the program. To our best knowledge, we do not know of any other system that is able to automatically perform such optimisations.

Our approach uses Multi-Objective Optimisation (MOO) and Genetic Programming (GP) [1]. In order to preserve semantic integrity whilst improving efficiency, we apply two sets of test cases. The first is co-evolved with the program population [2] to test the semantics of the programs. The second is drawn from a distribution modelling expected input, and is used to assess the non-functional properties of the code. The original function is used as an oracle to obtain the expected results of these test cases.

Evolving correct software from scratch is a difficult task [2], so we exploit the code of the input function by seeding the first generation of GP. The first generation will not be a random sample of the search space as is usually standard in GP applications, but it will contain genetic material taken from the original input function. Note that this approach is similar to our previous work on *Automatic Bug Fixing* [3], in which all the individuals of the first generation were equal to the original incorrect software, and the goal is to evolve a bugfree version. A similar approach has also been previously taken in attempting to reduce the size of existing software [4].

We present a preliminary implementation of the novel framework, and we validate it on a case study. We then apply systematic experimentation to determine the most important factors contributing to the success of the framework. Although our prototype is still in an early stage of development, this paper gives the important contribution of presenting a general method to automatically optimise code using evolutionary techniques. We are also able to provide some guidance to other practitioners in applying such an approach, based on our analysis of empirical results.

The paper is organised as follows. Section 2 describes in detail all the components of the novel framework, whereas Section 3 presents our case study. Section 4 describes our results and Section 5 suggests further work.

2 Evolutionary Framework

An overview of our framework is given in Figure 1. The framework takes as input the code of a function or program, along with an expected input distribution, and then it applies GP to optimise one or more non-functional criteria. Note that in our experimentation, we chose to parameterise the use of MOO and Co-evolution in order to assess their impact on the ability of the framework to optimise non-functional properties of the software. The main differences from previous GP work are how the first generation is seeded, how the training set is used and generated, the particular use of

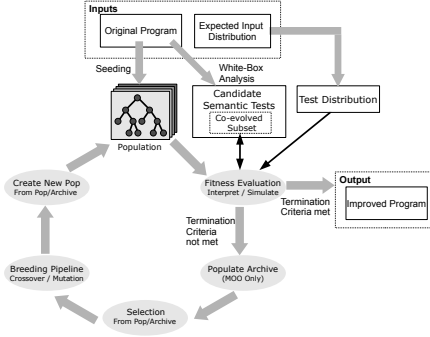


Fig. 1. Evolutionary Framework

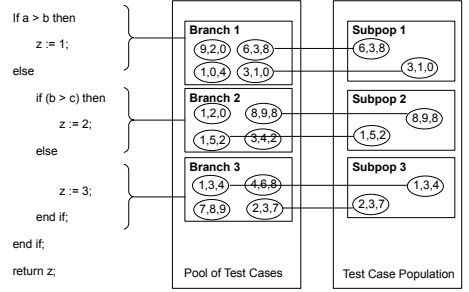


Fig. 2. The Relationship between a Program and the Semantic Test Set Population

multi-objective optimisation, and the employment of simulation and models in estimating non-functional properties of individuals.

2.1 Seeding Strategies

Usually, in GP applications the first generation is sampled at random, for example, using Koza’s ramped half-and-half initialisation method. Evolving bugfree software from scratch with GP is an hard task [5], but in our case we have as input the entire code of the function that we want to optimise, and we can exploit this information.

Different seeding strategies can be designed, and this is a case of the classic “exploration versus exploitation” trade-off that is so often an issue in heuristic search, and in particular evolutionary computation. On one hand, if we over-exploit the original program we might constrain the search in a particular sub-optimal area of the search space, i.e. the resulting programs will be very similar to the input one. On the other hand, ignoring the input genetic material would likely make the search too difficult. The point here is that, although we do not want a final program that is identical to the input one, its genetic material can be used as *building blocks* in evolving a better program. This has interesting implications for understanding how GP achieves its goal: *can building blocks be recombined in different ways to improve performance?*

In this work we consider a simple strategy: given a fraction δ of the initial random population, then δ individuals will be replaced by a copy of the input function. The remaining individuals are generated using a standard initialisation method.

2.2 Preserving Semantic Equivalence

Modifications to the input program can compromise its original semantics and our goal is to output an improved yet semantically equivalent program. It is important that our evaluation of individuals is effective in testing the semantics of new programs against the original. Exhaustive testing is usually impossible, and any testing strategy is therefore open to exploitation by an evolutionary algorithm through over-fitting.

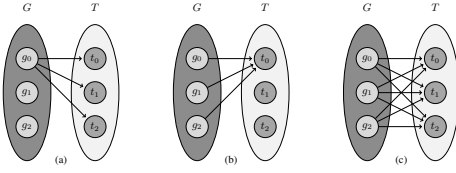


Fig. 3. G is the population of programs, whereas T is the population of test cases. (a) shows the test cases used to calculate the fitness of the first program g_0 . (b) shows the programs used to calculate the fitness of the first test case t_0 . Note the common arc between the first program and the first test case. Finally, picture (c) presents all possible $|G| \cdot |T|$ connections.

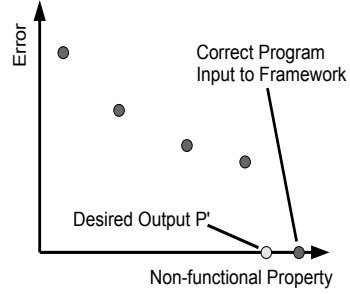


Fig. 4. A Pareto Front Composed of Five Programs in Objective Space

To improve the effectiveness of our fitness evaluation method, we employ a form of co-evolution, in principle similar to that used by Hillis [6]. Before the evolutionary algorithm begins, we first generate a large set of test cases using a White Box testing criterion [7], specifically *branch coverage*. This set is partitioned into subsets, one for each branch of the program. The partitioning ensures a degree of behavioural diversity amongst test cases.

The test set is then co-evolved as a separate population (the “training set”), from a selection from the larger pool produced prior to evolution. This training set is also partitioned, so that it samples each branch of the original program.

At each generation, the GP individuals are tested with the test cases in the training set. The sum of the errors from the expected results is referred to as the *semantic score* and is one component of the fitness of a GP individual. Figure 2 illustrates the relationships between the program and test set populations.

2.3 Evaluating Non-functional Criteria

To evaluate non-functional properties of individuals, a separate training set from that used to evaluate the semantic score is employed. The set is drawn from the expected input distribution provided to the framework, which could be based on probe measurement of software. For each non-functional criterion, a score is calculated for GP individuals using this set. The final fitness function of a GP individual will be composed of these scores and the semantic score. The set of tests is resampled at the start of each generation, to prevent overfitting of non-functional fitness for a particular set of inputs.

In this paper, we will estimate (by modelling and simulation) the number of CPU cycles consumed by each individual, assuming a uniform distribution of integer inputs for the case study. Note that this work is distinct from previous work on program compression [4] as the number of cycles used will depend on the path taken within a program. The framework can be extended to handle other types of non-functional criteria.

Simulation. The cycle usage of an individual can be estimated using a processor simulator and here we have used the M5 Simulator [8], targeted for an ARM Microprocessor.

The parameters of the simulator were left unchanged from their default values. Individuals are written out by the framework as C Code and compiled with an ARM-Targeted GCC cross-compiler. A single program is executed along with test code that executes the given test cases, and a total cycle usage estimate provided.

Whilst simulation does not perfectly reflect a physical system, it is worth noting that we are only concerned with *relative accuracy* between individuals, and also that the accuracy of simulation is an issue beyond the scope of our framework: we can easily incorporate alternatives or improvements.

Model Construction. Compiling and then testing each individual in a simulator can be computationally expensive. In this work we have carried out a large quantity of experiments as part of the analysis of the problem. Thus, we opted to study the approach of modelling the cycle usage as a linear model of instructions executed:

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots \beta_n x_n + \epsilon$$

Where Y is the estimated cycles consumed by a program, $x_1 \dots x_n$ are the frequencies that each of the n instructions appear within a program, and the coefficients $\beta_1 \dots \beta_n$ are an estimate of the cost of each instruction. ϵ is the noise term, introduced by factors not considered by the other components of the model. This is a simplification, because the ordering of the instructions affects the total cycles consumed due to pipelining and caching and because subsequent compiler optimisations will be dependent on the program structure.

To use such a model, the coefficients must be estimated. We achieved this by executing one large evolutionary run of the framework, and logging the frequencies with which each instruction appeared in each individual, and their corresponding cycle usage. Least Squares Linear Regression was then used to fit this model. It was possible to verify the relative accuracy of this model for the data points used in constructing it. As we are using tournament selection, we compared the results of using a model to carry out a tournament size 2 against the results of using the simulator results. The model was found to be in agreement with the simulator 65% of the time. It was not clear if this would be sufficient, and therefore the model was treated as a parameter of our experiments.

Using the Model. During experimentation, we execute the individuals through interpretation within the framework, whilst storing a profile of the nodes visited during evaluation. This profile is then used in conjunction with the model provided to the framework to estimate the number of the cycles the individual would consume. Thus a combination of interpretation and model-based estimation (or alternatively, simulation) can be used by the framework.

2.4 Multi-objective Optimisation

Our framework is faced with the challenge of optimising one or more non-functional properties, whilst retaining the correct semantics of the original program provided as input. This problem can be formulated as Multi-Objective Optimisation (MOO), indeed this is the approach we have taken previously to a similar problem [9]. In that work,

we were searching for a *set of possible trade-offs* between non-functional and function properties, but here we are concerned only with finding a *single program*. Hence it was unclear whether a fully MOO approach based on pareto-dominance was necessary.

We therefore adopted two approaches to combining objectives in fitness evaluation. The first was to use a simple linear combination of the functional and non-functional fitness measures. The second is to use the Strength Pareto Evolutionary Algorithm Two (SPEA2) [10]. This is a popular pareto-based method that attempts to approximate the pareto-front in objective space, as illustrated by Figure 4.

In Figure 4, it is assumed that the aim is to minimise both the non-functional property of the software and its error, that is both fitness components are *cost functions*. A pareto front would consist of the darker points, where no improvement in one objective can be made without worsening fitness in one of the other objectives. Our framework would like to find the point P' , a program with zero error and an improved non-functional fitness.

One possible justification of using a pareto-based MOO approach is the building block hypothesis often used to provide some rationale for genetic recombination in evolutionary algorithms. SPEA2 should find a set of programs that provide varying levels of error for different non-functional property values. Recombination between these smaller building blocks may produce re-orderings of instructions and new combinations that provide the same functionality but at a lower non-functional cost.

In our experimentation, we chose to make the MOO component of the framework a parameter, in order to establish what impact the two approaches would have on the success of the optimisation process.

3 Case Study

3.1 Software under Analysis

In our experiments, we analysed the *Triangle Classification* (TC) problem [7]. We choose that particular function because it is commonly used in the software testing literature, and the first one on which theoretical results have been obtained [11]. Given three integers as input, the output is a number representing whether the inputs can be classified as the sides of either an invalid, scalene, isosceles or equilateral triangle.

We used two different implementations, respectively published in [12] and [13] and expressed in Java in Figures 5 and 6 respectively. Note that their return values have been changed to make them consistent. The two implementations are not semantically equivalent, because they have faults related to arithmetic overflows.

3.2 Experimental Method

The framework was implemented in Java, and we used ECJ 16 [14] for the GP system. In particular, we used Strongly Typed Genetic Programming [15]. All the parameters of the framework that are not stated in paper have the default values in ECJ, as inherited from the `koza.params` parameter file.

For each TC version ($V1$ and $V2$) we carried out distinct experiments with 2 different cost models ($M1$ and $M2$), for a total of 4 independent sets of experiments. In

```

public int triangleClassification
(int a, int b, int c) {
    if (a > b) {int tmp = a; a = b; b = tmp;}
    if (a > c) {int tmp = a; a = c; c = tmp;}
    if (b > c) {int tmp = b; b = c; c = tmp;}
    if(a+b <= c)
        return 1;
    else {
        if(a == b && b == c) return 4;
        else if(a == b || b == c) return 3;
        else return 2;}
}

```

Fig. 5. 1st TC version [12]

```

public int triangleClassification
(int a, int b, int c) {
    if(a<=0 || b<=0 || c<=0) return 1;
    int tmp = 0;
    if(a==b) tmp += 1;
    if(a==c) tmp += 2;
    if(b==c) tmp += 3;
    if(tmp == 0){
        if((a+b<=c) || (b+c <=a) || (a+c<=b)) tmp = 1;
        else tmp = 2;
        return tmp;}
    if(tmp > 3) tmp = 4;
    else if(tmp==1 && (a+b>c)) tmp = 3;
    else if(tmp==2 && (a+c>b)) tmp = 3;
    else if(tmp==3 && (b+c>a)) tmp = 3;
    else tmp = 1;
    return tmp;
}

```

Fig. 6. 2nd TC version [13]

one model, each GP primitive has unitary estimated cycle cost ($M1$), whereas in the second model ($M2$) these costs have been estimated by least squares regression on data collected from a run using simulator.

For each group of experiments, we performed a full factorial design [16] of 8 parameters that we considered most important. Table 1 shows their high and low values. The total number of tested configurations was $4 \cdot 2^8 = 1024$. However, the SPEA2 archive is used only when MOO is employed, hence 256 experiments are redundant.

The probability that a tree is not affected by either crossover or mutation is 0.1; test case population size of 200, with an archive of 50 elements and a main pool of 2000 test cases; the cycle score is evaluated on 100 test cases that are sample at each generation with uniform distribution of values in $\{-127, \dots, 128\}$.

There are 36 GP primitives: 3 input variables, 1 other integer variable, read and write of variables, 1 variable wrapper, 10 integer constants, 5 arithmetic operators, 2 boolean constants, 8 boolean operators and 4 commands. There are 4 node return values: command, integer value, integer variable and boolean value.

Table 1. Factorial design of 8 parameters. Note that $P_m = 0.9 - P_c$ and $S \cdot G = 50000$ such that the total number of fitness evaluations remains constant. For the same reason, when MOO is employed, the population is reduced by the SPEA2 archive size. If co-evolution is not employed, the test cases are simply sampled at random at each generation. If MOO is not employed, the semantic and the cycle scores are linearly combined, with a weight of 128 for the semantic score. A mutation event is a single mutation from a pool of ECJ mutation operators is applied.

Parameter	Id	Low Value	High Value
Probability of Crossover (P_c)/Mutation(P_m)	X1	0.1/0.8	0.8/0.1
Population Size (S) and Generations (G)	X2	50/1000	1000/50
Tournament Selection Size	X3	2	7
Types of Mutations	X4	1	6
Clone Proportion δ	X5	0	1
Co-evolution Enabled	X6	false	true
SPEA2 MOO Used	X7	false	true
SPEA2 Archive Proportion	X8	$\frac{1}{9}$	1

Table 2. P-values of the ANOVA tests run on the 4 different types of experiments

Configuration	X1	X2	X3	X4	X5	X6	X7	X8
V1 M1	0.0001	0	0	0.3396	0	0	0	0
V1 M2	0	0.0816	0.2707	0	0	0	0	0.8529
V2 M1	0.0026	0	0	0.2094	0	0	0	0
V2 M2	0	0	0	0	0	0	0	0.7858

Table 3. For each of the 4 configurations, the parameter settings that result in the highest average and max gain scores are reported, as well as their best performance gains

Config.	X1	X2	X3	X4	X5	X6	X7	X8	Average	Max	Variance	Best
V1 M1	0.8/0.1	1000/50	2	6	1	true	false	-	1330.7	1826.0	163,700	7355
	0.8/0.1	50/1000	7	1	1	true	false	-	1052.7	2507.0	305,100	11610
V1 M2	0.1/0.8	50/1000	7	1	1	true	false	-	175.8	3402.9	2,211,000	-1618
	0.1/0.8	50/1000	7	6	1	true	false	-	-1528.2	3609.5	1,991,800	-503
V2 M1	0.1/0.8	1000/50	7	6	1	true	false	-	570.3	1475.0	181,600	6645
	0.8/0.1	50/1000	7	6	1	true	false	-	237.4	1519.0	105,700	6645
V2 M2	0.1/0.8	1000/50	2	1	1	true	true	1	39.3	1490.4	77,700	-2198
	0.8/0.1	50/1000	7	1	1	true	false	-	-318.0	5401.1	2,688,700	-11242

If P is the program given as output by the framework, we are interested whether P is faster than the input program. Given an output, we validate P against an independent set of 10,000 test cases. If P fails any of those test cases, the framework has failed to produce a semantically equivalent program, and P will be replaced by the original program. Note that passing 10,000 test cases does not guarantee the equivalence of semantics, so the output programs need to be manually checked at the end of a run.

The performance of P is evaluated with the *gain score*, that is the difference of the cycle scores of the original program and P . These cycle scores are evaluated on 100 test cases. The faster P is, the higher gain score it will receive. If P is not correct, then the gain score is 0. It is possible that the gain score assumes a negative value.

For each of the 1024 configurations, we ran them 100 times and recorded the gain score. For each of the 4 groups of experiments we report an ANOVA analysis of the results in Table 2, whereas the configurations that gives the highest single and average gain score are reported in Table 3. Moreover, for each best configuration in Table 3 we chose the best program (out of 100 trials), and we evaluated the estimated real gain score by running it in the simulator against the original program (on 1,000 input triplets over the expected input distribution).

4 Results and Conclusions

Table 2 demonstrates that the design decisions made in selecting each parameter value have a significant impact (i.e. have a small p-value) on the performance improvement achieved. Only X3, X4 and X8 are significant for only part of the experiments. All are concerned with the amount of exploration the search performs, and it is conjectured that the significance of these parameters will be problem-specific.

The best individual data in Table 3 shows that improvements were possible for both original programs. Model 2, based on the simple linear model building approach, performed poorly: we recommend that hand-crafted models of resource usage or full simulation should be used.

Seeding the initial population based on the original program is a useful technique that should be used. Similarly, the application of co-evolution is an effective measure to improve performance.

When employing GP in general, a large population for a smaller number of generations is usually more effective than a smaller one evolved over a large number of generations, due to the prevalent problem of bloat [17]. However, in our experiments we see exactly the opposite trend where small populations are more successful in that they produce the largest improvements in program speed. It would usually be expected that a higher number of generations tends to lead to over-fitting and fewer correct programs over a succession of runs. However, the efficiency gains are best in the very few cases in which the resulting programs are actually correct.

The fact that a pareto-based MOO approach mostly provides worse results may be due to the fact that the programs we analysed in our case study can be optimised to some extent using multiple mutations, and with few changes in the source code. Hence a search concentrated around the input program gives better results, rather than spread across a range of program shapes and sizes. It is therefore possible that pareto-based MOO will find superior solutions than a linear combination given more resources.

It is worth noting that for the non-binary parameters (e.g., crossover rate) we analysed only low and high values in our experiments. The best tunings likely lie within those extremes and it is likely that better results than in Table 3 could be obtained by tuning these parameters.

5 Summary and Future Work

In this paper we have presented a novel framework for improving non-functional criteria of software. The framework has been successfully used for evolving new correct and faster versions of the programs in our case study. Regarding the quality of the final outcomes, the experiments also showed expected and unexpected roles of some parameter settings.

Immediate future work is to test if these results hold for other problems. We would also like to further investigate optimal parameter settings, in particular the cloning proportion used. Also, alternative seeding strategies could be investigated, potentially as an opportunity to investigate GP schema theory [17] where seeding according to schemas may have a beneficial effect.

As already discussed, further work using MOO and extended evolutionary runs may allow us to provide more guidance on parameter selection.

Acknowledgements

The authors are grateful to Simon Poulding, Per Kristian Lehre and Ramón Sagarna for insightful discussions. This work is supported by an EPSRC grant (EP/D052785/1).

References

1. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
2. Arcuri, A., Yao, X.: Coevolving programs and unit tests from their specification. In: IEEE International Conference on Automated Software Engineering (ASE), pp. 397–400 (2007)
3. Arcuri, A., Yao, X.: A novel co-evolutionary approach to automatic software bug fixing. In: IEEE Congress on Evolutionary Computation (CEC), pp. 162–168 (2008)
4. Langdon, W.B., Nordin, P.: Seeding genetic programming populations. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) EuroGP 2000. LNCS, vol. 1802, pp. 304–315. Springer, Heidelberg (2000)
5. Reformat, M., Xinwei, C., Miller, J.: On the possibilities of (pseudo-) software cloning from external interactions. *Soft Computing* 12(1), 29–49 (2007)
6. Hillis, W.D.: Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D* 42(1-3), 228–234 (1990)
7. Myers, G.: The Art of Software Testing. Wiley, New York (1979)
8. Binkert, N., Dreslinski, R., Hsu, L., Lim, K., Saidi, A., Reinhardt, S.: The M5 simulator: Modeling networked systems. *IEEE Micro*. 26(4), 52–60 (2006)
9. White, D.R., Clark, J., Jacob, J., Poulding, S.: Searching for Resource-Efficient Programs: Low-Power Pseudorandom Number Generators. In: GECCO 2008: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, pp. 1775–1782 (2008)
10. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Swiss Federal Institute of Technology (2001)
11. Arcuri, A., Lehre, P.K., Yao, X.: Theoretical runtime analyses of search algorithms on the test data generation for the triangle classification problem. In: International Workshop on Search-Based Software Testing (SBST), pp. 161–169 (2008)
12. McMinn, P.: Search-based software test data generation: A survey. *Software Testing, Verification and Reliability* 14(2), 105–156 (2004)
13. Miller, J., Reformat, M., Zhang, H.: Automatic test data generation using genetic algorithm and program dependence graphs. *Info. and Software Technology* 48(7), 586–605 (2006)
14. ECJ: Evolutionary computation in Java, <http://www.cs.gmu.edu/~eclab/projects/ecj/>
15. Montana, D.J.: Strongly typed GP. *Evolutionary Computation* 3(2), 199–230 (1995)
16. Montgomery, D.C.: Design and Analysis of Experiments. John Wiley & Sons, Chichester (2006)
17. Langdon, W.B., Poli, R.: Foundations of Genetic Programming. Springer, Heidelberg (2002)

Policy Evolution with Grammatical Evolution^{*}

Yow Tzu Lim¹, Pau Chen Cheng², John Andrew Clark¹, and Pankaj Rohatgi²

¹ Department of Computer Science, University of York, UK

{yowtzu,jac}@cs.york.ac.uk

² Department of Security and Privacy, IBM T J Watson Research Center, USA

{pau,rohatgi}@us.ibm.com

Abstract. Security policies are becoming more sophisticated. Operational forces will often be faced with making tricky risk decisions and policies must be flexible enough to allow appropriate actions to be facilitated. Access requests are no longer simple subject access object matters. There is often a great deal of context to be taken into account. Most security work is couched in terms of risk management, but the benefits of actions will need to be taken into account too. In some cases it may not be clear what the policy should be. People are often better at dealing with specific examples than producing general rules. In this paper we investigate the use of Grammatical Evolution (GE) to attempt to infer Fuzzy MLS policy from decision examples. This approach couches policy inference as a search for a policy that is most consistent with the supplied examples set. The results show this approach is promising.

1 Introduction

In computer systems, a security policy is essentially a set of rules specifying the way to secure a system for the present and the *future*. Forming a security policy is a challenging task: the system may be inherently complex with many potentially conflicting factors. Traditionally security policies have had a strong tendency to encode a static view of risk and how it should be managed (most typically in a pessimistic or conservative way) [1]. Such an approach will not suffice for many dynamic systems which operate in highly uncertain, inherently risky environments. In many military operations, for example, we cannot expect to predict all possible situations.

Much security work is couched in terms of risk but in the real world there are benefits to be had. In military operations you may be prepared to risk a

^{*} Research was sponsored by US Army Research laboratory and the UK Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

compromise of confidentiality if not doing so could cost lives. There is a need for operational flexibility in decision making, yet we cannot allow recklessness. Decisions need to be defensible and so must be made on some principled basis. People are typically better at making specific decisions than in providing abstract justification for their decisions. It is very useful to be able to codify in what a “principled basis” consists since this serves to document “good practice” and facilitates its propagation.

The above discussion has been couched in terms of human decision making. In some environments the required speed of system response may force an automated decision. Such automated decisions must also be made on a “principled basis”, and some of these decisions may be very tricky. Automated support must be provided with decision strategies or rules to apply.

In this paper we investigate how security policy rules can be extracted automatically from examples of decisions made in specified circumstances. This is an exercise in *policy inference*. The automation aspect of the inference is doubly useful: automated inference techniques can discover rules that humans would miss; and policies can be dynamically inferred as new examples of tricky decisions become available. Thus the current policy can evolve to reflect the experience of the system. For example, if a human determines what the proper response should be based upon the information available, either in real-time or post facto, a conclusion is drawn that similar responses should be given under similar circumstances. Essentially, we attempt to partition the decision space; each partition is associated with a response that is commensurate with the risk vs. benefit trade-off.

In practice, decision makers may come to different decisions in the same circumstances, particularly if the decisions are tricky. Decision makers may use data that is not available to the inference engine to reach a decision, or else decision makers may simply have different appetites for risk. Any inference technique must be able to handle decision examples that do not seem to be entirely consistent. The chosen inference approach here is Grammatical Evolution (GE).

The organisation of this paper is as follows: Section 2 discusses the related work. Section 3 introduces Grammatical Evolution. Section 4 presents the Fuzzy MLS policy model, which serves as the learning target throughout the paper. Sections 5 and 6 present the experimental setup and results. Finally Section 7 concludes the paper and suggests possible future work.

2 Related Work

The application of machine learning techniques on security domain has been focused on intrusion and anomaly detections. Machine learning techniques are used to discover useful patterns of system features that describe the system behaviour. Then, these features are used to recognise anomalies and intrusions. For detail, refer [2]. However, there are limited known attempts in generating the policy automatically from previous decision examples using machine learning techniques. In [3] autonomic security policies were mentioned yet no results have

appeared. The only published works are found in [4] in which Genetic Programming is used to infer various Fuzzy MLS policy model from decision examples. The results show GP based approach is promising.

On the other hand, rule inferencing techniques have been around for many years in machine learning domain. Emphasis in this paper is placed on Grammatical Evolution (GE). GE has emerged in recent years as a promising technique. A list of applications and other GE papers can be found at <http://www.grammatical-evolution.org/pubs.html>. In [5,6] GE is used to learn investment strategy (trading rules) using the market stock indices. The results show the investment strategy inferred yields profitable performance for the trading periods analysed. In [7] GE is used to evolve rules for foreign exchanges. The rules learnt outperforms the benchmark buy and hold strategy over all the trading periods analysed. However, there have been no previous known research on GE in our domain of interest — security policy.

3 Grammatical Evolution

Grammatical Evolution (GE) is an evolutionary approach to automated programming. In GE, individuals are variable length binary strings. These individuals serve as indices that map a set Backus Naur Form (BNF) language grammar to programs. These programs can then be executed for fitness evaluation purposes. To solve a problem using GE, the language grammar must first be specified. In BNF, the grammar of a language is represented as a tuple $\{N, T, S, P\}$ [8] where N is the non-terminal set, T is the terminal set, S is a special element of N called start symbol which is expanded first. P is the set of production rules (also called derivation rules) that is used to guide the expansion from N to T . For example, the following grammar defines a set of arithmetic expressions over the variables X and Y [8]:

$$\begin{aligned} N &= \{\langle \text{expr} \rangle, \langle \text{op} \rangle, \langle \text{var} \rangle\} \\ T &= \{+, -, *, /, X, Y, (,)\} \\ S &= \langle \text{expr} \rangle \end{aligned}$$

P consists of a set of production rules as follows:

```

<expr> ::= <expr><op><expr> | (<expr><op><expr>) | <var>
<op>   ::= + | - | / | *
<var>  ::= X | Y

```

The mapping process between the variable length binary individuals and programs using grammar is known as genotype-phenotype mapping. This process is best illustrated with an example. In each individual (genome), every 8 consecutive bits is viewed as an integer which are more commonly known as codon. Consider an individual with the following codons:

[10, 5, 51, 8, 16, 49, 30, 18]

The codon value is read to determine the usage of a certain production rule to expand the symbol using a modulus mapping function as follows:

$$Rule = Codon\ value \% Total\ number\ of\ production\ rules$$

where % is the arithmetic modulus operator. Assuming the grammar used is the arithmetic expression grammar discussed, the start symbol is $\langle expr \rangle$ and the first codon value is 10. $10 \% 3 = 1$. The production rule 1 is used; $\langle expr \rangle$ is replaced by $(\langle expr \rangle \langle op \rangle \langle expr \rangle)$. Assuming leftmost derivation is used, we expand the left most $\langle expr \rangle$ after '(' which is a terminal. As the second codon value is 5 and the production rules available is 3. $5 \% 3 = 2$. The production rule 2 is used; the leftmost $\langle expr \rangle$ is replaced by $\langle var \rangle$ resulting $(\langle var \rangle \langle op \rangle \langle expr \rangle)$. The process continues until a complete program is generated, i.e. when all the non-terminals have been mapped to terminals.

Some problems can arise during the mapping process. Firstly, the codon values may run out before the entire mapping process is complete. To overcome this, individual can be wrapped around to reuse the codon value. Second, the mapping process also can be indefinitely long if the grammar used is recursive. This can be resolved by setting an upper bound on the number of times the production rules expansion that can be performed. One possible way to do this is to set a threshold limit on the number of times an individual can wraparound. If a complete program is not fully generated when this threshold is reached, the individual is assigned the lowest possible fitness value.

The greatest advantage of GE is that it allows artifacts to be evolved in a grammar compliant way and many solution spaces can be defined using grammars. The use of grammar also makes program generation in arbitrary languages possible by only changing the BNF grammar. The search algorithm is also an independent component in the system. This feature allows GE to reap the advantages of any improvement in any evolutionary algorithm.

4 Fuzzy MLS Model

Fuzzy MLS policy model [9] is an adaptive extension to traditional MLS (multi-level security) Bell-LaPadula policy model [10]. In Bell-LaPadula policy model, every subject and object is assigned a security label ($\langle$ sensitivity level, categories set $\rangle$). For a read access, r , the policy can be summarised as follows:

$$\begin{aligned} & \text{IF } sl \geq ol \text{ AND } sc \supseteq oc \text{ THEN } r \text{ is allowed} \\ & \text{IF } sl < ol \text{ OR } sc \not\supseteq oc \text{ THEN } r \text{ is denied} \end{aligned} \quad (1)$$

where sl and ol are subject and object sensitivity levels and sc and oc are subject and object category sets. In other words, a subject can access an object iff the subject is trustworthy enough ($sl \geq ol$) and has the legitimate “need-to-know” ($sc \supseteq oc$) to access the object. In terms of risk, the traditional MLS policy can be viewed as a fixed trade-off setting between risk of information disclosure versus the benefit an organisation can gain from it.

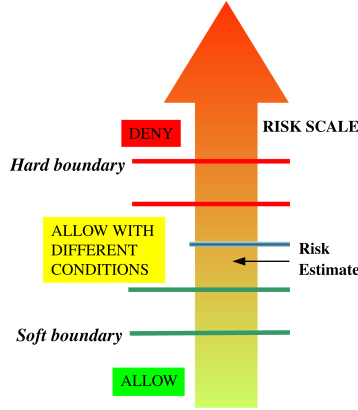


Fig. 1. Risk adaptive access control on a risk scale

In Fuzzy MLS model, the *quantified estimates* of the risk for the *unauthorised disclosure of information* are computed. Then, these estimates are used to build a risk scale shown in Figure 1.

The risk scale is divided into multiple bands; each band is associated with a decision. The risk in the bottom band is considered low enough so the decision is simply *allow* whereas the risk in the top band is considered too high so the decision is *deny*. Each band between the top and bottom is associated with a decision *allow with risk mitigation measures*.

Fuzzy MLS model defines risk as the *expected value of damage* caused by unauthorised disclosure of information:

$$risk = (expected\ damage) \times (probability\ of\ unauthorised\ disclosure)$$

The value of the damage is estimated from the object’s sensitivity level. The probability of unauthorised disclosure is estimated by quantifying two “gaps”: one between the subject’s and the object’s sensitivity levels and the other between the subject’s and the object’s category sets. For simplicity, this experiment looks only at the sensitivity levels and assumes the categories sets are the same, thus risk becomes a function of subject and object sensitivity levels only. For more detail on risk quantification, see [9].

A function that maps an estimated risk to a risk band is defined in [4] as follow:

$$band(risk(sl, ol)) = \min(\lfloor \log_{10}(risk(sl, ol)) \rfloor, N - 1) \quad (2)$$

The band numbers start from 0, N is the number of bands desired on the scale and the function $risk(sl, ol)$ is defined in [9]. The intuition of using base-10 logarithm in (2) is to use a risk band number to represent *the order of magnitude* of risk. Since each band is associated with a decision, a risk band number computed using (2) represents a possible decision in the policy.

5 Experiments

A policy can be viewed as a function which maps decision making factors to a decision. In Fuzzy MLS model, this mapping is the composition of the band and risk functions shown in (2). GE is used in this experiment to search for an equivalence of this composition function using a set of decision examples. Each individual represents a candidate function. The search engine used is steady state genetic algorithm with 1% replacement rate scheme used and the selection is Roulette Wheel Selection. The set of production rules, P used is:

```

<expr>      ::= <unary_op>(<expr>)|<bin_op>(<expr>,<expr>)|<sub_expr>
<sub_expr>  ::= <var>|<const>
<unary_op>  ::= sin|cos|exp|protectedLog10|ceil|floor|exp|-
<bin_op>    ::= add|minus|multiply|protectedDiv|min|max
<var>       ::= s1|o1
<const>     ::= 0.<digit><digit>
<digit>     ::= 0|1|2|3|4|5|6|7|8|9

```

The angle brackets distinguish the non-terminal symbols, N from terminal symbol, T . The starting symbols is $\langle \text{expr} \rangle$.

The primitive operators are wrapped as a function call to prevent any bias being introduced among the operators. All functions work in accordance to the specification defined in the ANSI C standard `<math.h>` library except the following four: $\min(x, y)$, $\max(x, y)$, $\text{protectedDiv}(x, y)$ and $\text{protectedLog10}(x)$. $\min$ and $\max$ operators return the minimum and maximum values between x and y , $\text{protectedDiv}(x, y)$ returns x / y if $y \neq 0$ and 1 if $y = 0$, and $\text{protectedLog10}(x)$ returns $\log_{10}(x)$ if $x > 0$ and 1 otherwise. Instead of using ephemeral random constant (ERC), the `const` and `digit` rules are used to generate random constant in the range of $(-1, 1)$. Generating random constant in this fashion enables random number to partake in the evolutionary process.

Crossover and mutation operations are applied to individual at each iteration with probability of 0.9 and 0.01 respectively. The mutation operator used is the standard bit-level GE mutation which flips each bit of the individual probabilistically. Two experiments have been carried out to investigate two different implementation of crossover operations; one-point crossover and effective crossover. One point crossover first chooses a point randomly on each of the two selected individuals and all data beyond that point are swapped between the two individuals. The chosen crossover point may happen to be after the effective length of the individual (the portion of the individual that are actually used to select the rules) and render the crossover operation ineffective. Effective crossover restricts the chosen crossover point in the range of effective length.

The initial population size is set to be 1024. Two experiments have been carried out to investigate two population initialisation methods. In the first experiment, each individual is initialised randomly with lengths in the range of [15, 25]. In the second experiment, the sensible initialisation method [11] is used. This initialisation method models the popular *ramp half and half* initialisation

method [12]; it takes the population size, minimum and maximum heights of the derivative trees permitted in the population as input parameters and generates approximately 50% full derivative trees with maximum height while the 50% have heights between the minimum and the maximum heights. In this experiment, the minimum and maximum heights are set to be 1 and 10.

The fitness evaluation for each individual is evaluated against a set of training examples. Each example x is a tuple of $(sl_x, ol_x, band_x)$, where $band_x$ is calculated using (2) and therefore all the examples used are correct. The training set consists of all 100 (sl, ol) integer pairs for sl and ol in the range of $[0, 9]$.

Two experiments have been carried out to investigate two different fitness functions. In the first experiment, the weighted fitness function (3) presented in [13] is used. This allows us to compare the performances of GE and GP side by side. Two principles are used to determine the score for a decision made by an individual. For an example x and an individual i , if i evaluates x to be in band j (j is $eval(i, x)$ rounded to the nearest integer), then:

- For a correct decision, *reward more* the higher the risk band,
- For an incorrect decision, *punish more* the more the decision deviates from the target. Also, punish *under-estimation* of the risk band more than over-estimation of it, i.e., punish more if $band_x > j$.

Based upon these principles, $fitness(i)$ is defined as follow:

$$fitness(i) = \sum_{\forall \text{ example } x} score(x) \quad (3)$$

where

$$score(x) \begin{cases} band_x + 1 & \text{if } j \equiv band_x, \\ -(band_x - j) & \text{if } band_x > j, \\ -(j - band_x)/2 & \text{if } band_x < j \end{cases}$$

The fitness function in the second experiment is inversely proportional to the sum of absolute differences between the value an individual evaluated to with the correct band encoded in the examples:

$$fitness(i) = \frac{1}{1 + \sum_{\forall \text{ example } x} |round(eval(i, x)) - band_x|} \quad (4)$$

6 Result and Discussion

Each experiment described is run 10 times. Two testing sets are used to evaluate the performance of the best individual. The first testing set is same as the first 100-example training set. This testing set provides a good indication on how much “knowledge” has been acquired by the approach employed in a fixed number of generations. The second testing set consists of 100 randomly generated

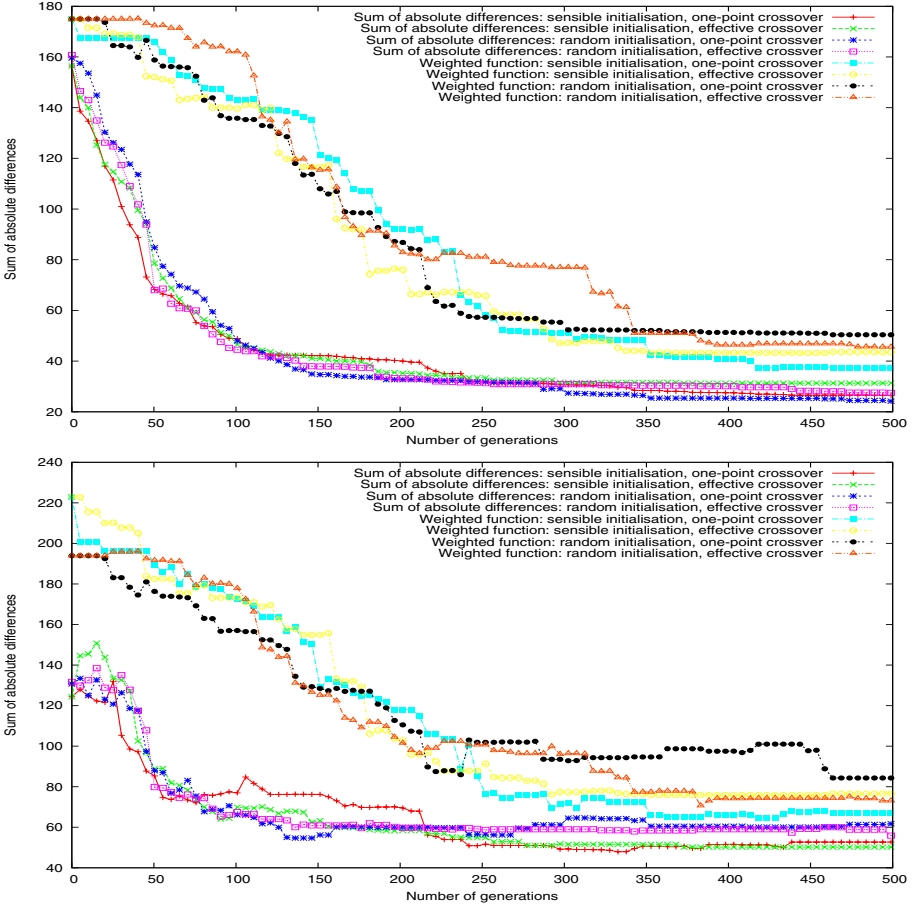


Fig. 2. The average sum of absolute differences between the bands output by the fittest individual and the correct bands in the 100 examples in Testing Set 1 and 2

(sl , ol) pairs where sl and ol are *real numbers* in $[0.0, 9.0]$. Therefore, most of these examples are unseen yet similar to training examples. This set provides a good measure on how much the acquired knowledge can be applied for unseen cases. The performance in terms of the sum of differences between the bands output by the fittest individual and the correct bands in the 100 examples in the testing set is shown in Figure 2 respectively.

The results show that the policies evolved using sum of absolute differences and outperform the ones evolved using weighted fitness function in both cases at all time during evolution. The sum of absolute differences fitness function provides faster learning speed (steeper slope); it only requires ≈ 100 generations to has the population to become stable as opposed to ≈ 250 generations required by the weighted fitness function. However, the uses of effective crossover and sensible initialisation do not provide any significant performance gain.

Table 1. Performance comparison between policies inferred using GP and GE with weighted fitness function (3)

Experiment	Testing set	Difference from target band				Mean difference
		0	1	2	≥ 3	
GP	1	62.3	9.6	6.0	22.1	1.248
	2	43.6	20.7	7.3	28.4	1.703
GE (SI ¹ , EC ³)	1	78.0	11.8	4.1	6.1	0.435
	2	47.7	39.5	7.3	5.5	0.766
GE (SI ¹ , OC ⁴)	1	80.5	10.6	3.4	5.5	0.373
	2	51.4	37.0	7.0	4.6	0.672
GE (RI ² , EC ³)	1	78.7	10.7	4.2	6.4	0.456
	2	49.8	38.4	6.7	5.1	0.731
GE (RI ² , OC ⁴)	1	76.6	11.6	4.4	7.4	0.504
	2	47.0	38.6	7.5	6.9	0.843

¹ Sensible Initialisation. ² Random Initialisation.³ Effective Crossover. ⁴ One-point Crossover.

In comparison to the results obtained using Genetic Programming with the same weighted fitness function and similar parameter settings [13], the performance of GE is superior over GP in terms number of correct bands and mean difference from target band in all cases. The results are summarised in Table 1. The comparison of results using sum of absolute differences fitness function cannot be done because there is no equivalent experiment carried out in GP.

Lastly, the size of the best individuals are much smaller compared to those in GP. This can be due to the fact that conforming to the grammar correctness in GE is relatively harder than conforming to the type correctness in GP.

7 Conclusion

We investigated the possibility of using Grammatical Evolution (GE) to infer Fuzzy MLS policy from examples. The results show that this approach is promising. The policies inferred using GE are found to outperform the ones inferred using GP with similar settings [13]. The policies inferred using sum of absolute differences fitness function is found to outperform the ones inferred using weighted fitness function. However, the uses of sensible initialisation and effective crossover in experiment do not provide any significant performance gain.

The availability of a diverse training set currently used in the experiment is difficult to obtain in real-life scenario. Some envisaged work includes: the use of a “less ideal” training set, possibly with the inclusion of wrong examples and skewed distribution on the examples set. One possible solution is to introduce fuzzy set concept. This approach is found to be able to increase the accuracy performance and more resilient to missing examples in the training set [13].

Inferring security policy with evolutionary algorithm is a very interesting domain. Everyone accepts that policy specification is currently hard, and things are set to worsen as systems are deployed in ever more complex environments with increasing sophistication and subtlety of decision making needed. This work

shows GE has very considerable promise. We recommend this important application area to fellow researchers.

References

1. Horizontal Integration: Broader Access Models for Realizing Information Dominance. Technical Report JSR-04-132, The MITRE Corporation JASON Program Office, Mclean, Virginia (December 2004)
2. Barbara, D.: Applications of Data Mining in Computer Security. Kluwer Academic Publishers, Norwell (2002)
3. McDaniel, P.D.: Policy Evolution: Autonomic Environmental Security (December 2004)
4. Lim, Y.T., Cheng, P.C., Clark, J.A., Rohatgi, P.: Policy Evolution with Genetic Programming. Technical report, IBM Research Report RC24442 (2008)
5. Dempsey, I., O'Neill, M., Brabazon, A.: Adaptive Trading with Grammatical Evolution. In: Proceedings of the 2006 IEEE Congress on Evolutionary Computation, Vancouver, July 6-21, 2006, pp. 9137–9142. IEEE Press, Los Alamitos (2006)
6. Brabazon, T., O'Neill, M., Ryan, C., Collins, J.J.: Uncovering Technical Trading Rules Using Evolutionary Automatic Programming. In: Proceedings of 2001 AAANZ Conference (Accounting Association of Australia and NZ), Auckland, New Zealand, July 1-3 (2001)
7. Brabazon, T., O'Neill, M.: Trading Foreign Exchange Markets Using Evolutionary Automatic Programming. In: Barry, A.M. (ed.) GECCO 2002: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference, New York, July 8, 2002, pp. 133–136. AAAI, Menlo Park (2002)
8. O'Neill, M., Ryan, C.: Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language. Genetic programming, vol. 4. Kluwer Academic Publishers, Dordrecht (2003)
9. Cheng, P.C., Rohatgi, P., Keser, C., Karger, P.A., Wagner, G.M., Reninger, A.S.: Fuzzy Multi-Level Security: An Experiment on Quantified Risk-Adaptive Access Control. In: IEEE Symposium on Security and Privacy, pp. 222–230 (2007)
10. Bell, D.E., LaPadula, L.J.: Computer Security Model: Unified Exposition and Multics Interpretation. Technical Report ESD-TR-75-306, The MITRE Corporation, Bedford, MA. HQ Electronic Systems Division, Hanscom AFB, MA (March 1976)
11. Ryan, C., Azad, R.M.A.: Sensible Initialisation in Chorus. In: Ryan, C., Soule, T., Keijzer, M., Tsang, E., Poli, R., Costa, E. (eds.) EuroGP 2003. LNCS, vol. 2610, pp. 394–403. Springer, Heidelberg (2003)
12. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
13. Lim, Y.T., Cheng, P.C., Clark, J.A., Rohatgi, P.: Policy Evolution with Genetic Programming: a Comparison of Three Approaches. In: 2008 IEEE Congress on Evolutionary Computation, Hong Kong, IEEE Computational Intelligence Society, June 1-6, 2008, pp. 813–819. IEEE Press, Los Alamitos (2008)

A PSO Based Adaboost Approach to Object Detection

Ammar W. Mohemmed, Mengjie Zhang, and Mark Johnston

School of Mathematics, Statistics and Computer Science
Victoria University of Wellington, P.O. 600, Wellington, New Zealand
{`ammam.mohemmed, mengjie.zhang, mark.johnston`}@mcs.vuw.ac.nz

Abstract. This paper describes a new approach using particle swarm optimisation (PSO) within AdaBoost for object detection. Instead of using the time consuming exhaustive search for finding good features to be used for constructing weak classifiers in AdaBoost, we propose two PSO based methods in this paper. The first uses PSO to evolve and select the good features only and the weak classifiers use a kind of decision stump. The second uses PSO for both selecting the good features and evolving weak classifiers in parallel. These two methods are examined and compared on a pasta detection data set. The experiment results show that both approaches perform quite well for the pasta detection problem, and that using PSO for selecting good individual features and evolving associated weak classifiers in AdaBoost is more effective than for selecting features only for this problem.

1 Introduction

Object detection attempts to determine the existence of specific objects in a set of images and, if present, to determine the locations, sizes and shapes of these objects. It is a challenging problem because objects can occur under different poses, lighting conditions, backgrounds and clutter. It often utilises a trained binary classifier/detector that can distinguish the objects of interest from the background (including objects of other classes).

One of the methods that was intensively investigated to improve the performance of object detection/classification is to use an ensemble of classifiers emulating the advantage of the “power of a group”. Instead of attempting to build a single (strong) classifier, a bundle of classifiers that individually are not necessarily powerful, are grouped to share the burden of the classification/detection task. Studies have shown that the performance of the ensemble is better than any of its components acting alone [1]. A large number of combination schemes and ensemble methods have been proposed in literature (for a survey see [2]), which can be categorised into two approaches. The first approach is the use of an ensemble of accurate, well trained classifier members. The effectiveness of this approach depends on the accuracy and diversity of the members [3,4]. To achieve good performance, the individual members in the ensemble should exhibit low error rates and produce uncorrelated errors.

The second approach to ensemble classification is to allow more tolerance to the accuracy of the individual classifiers, which are used as *weak* classifiers for making an assembly [5]. Two popular methods are Bagging and Boosting, which both rely on re-sampling the features to obtain different training sets for each of the classifiers. Bagging [6] is based on combining classifiers each individually trained on a *bootstrap replica* of the original training set. Boosting refers to a general and provably effective method of producing an accurate ensemble by combining rough and moderately inaccurate rules of thumb. The idea of Boosting has its roots in PAC (Probably Approximately Correct) learning [7]. Kearns and Valiant [8] proved the fact that learners, each performing only slightly better than random, can be combined to form an arbitrarily good ensemble hypothesis [9].

Adaboost is an “adaptive boost” algorithm [10] that solves many of the practical difficulties of the early boosting algorithms. A major idea is to maintain a distribution (set of weights) over the training set. All weights are initially set equally, but at each iteration, the weights of incorrectly classified examples are increased so that the weak learner is forced to focus on the hard examples in the training set. Viola and Jones [11] make a successful application of Adaboost to face and pedestrian detection. The algorithm builds an ensemble of weak classifiers with each classifier based on a single feature called Haar-like features. There are typically hundreds of thousands of features available and Adaboost uses an exhaustive search to find the set of best features for discriminating between positive and negative examples. So this exhaustive search usually involves a very high computational cost for finding the best features for use of the weak classifiers. Another limitation of Adaboost is that it does not specify how to train a weak classifier and it is left to the practitioner to decide the nature of classifier.

1.1 Goals

To avoid these limitations, the goal of this paper is to investigate a new approach using PSO within AdaBoost for object detection. Instead of using an exhaustive search, we aim to use PSO to automatically search for good features to reduce the computational cost. We will consider two PSO approaches for this purpose. The first one (*AdaBoostPSO1*) considers using PSO to evolve and select the good features only and the weak classifiers use a kind of decision stump. The second (*AdaBoostPSO2*) considers using PSO for both selecting the good features and evolving weak classifiers parallelly. These approaches will be examined and compared on a pasta detection data set, which was derived from the competition at GECCO 2006. Specifically, we investigate:

- how PSO can be used for selecting good features to be used by the weak classifiers in AdaBoost;
- whether the two PSO approaches can achieve acceptable results on the pasta data set; and
- whether AdaBoostPSO2 outperforms AdaBoostPSO1 on this problem.

In the remainder of the paper, Section 2 briefly describes the background about Adaboost for object detection and PSO. Section 3 describe the new

PSO approaches for selecting good features and evolving weak classifiers. After presenting the experiment design in Section 4, Section 5 describes the experimental results with discussion. Section 6 gives conclusions and future work directions.

2 Background

2.1 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a population based stochastic optimization tool inspired by social behaviour of flocks of birds (and schools of fish etc.), as developed by Kennedy and Eberhart in 1995 [12]. PSO starts with a population of particles whose positions represent the potential solutions for the studied problem and velocities are randomly initialized in the search space. The search for an optimal position (solution) is performed by updating the velocity and position of particle i according to the following two equations:

$$V_{id} = V_{id} + \phi_1 r_1 (B_{id} - X_{id}) + \phi_2 r_2 (B_{id}^n - X_{id}) \quad (1)$$

$$i = 1, 2, \dots, N_s, \text{ and } d = 1, 2, \dots, D$$

$$X_{id} = X_{id} + V_{id} \quad (2)$$

where ϕ_1 and ϕ_2 are positive constants, called *acceleration coefficients*, N_s is the total number of particles in the swarm, D is the dimension of problem search space, i.e., number of parameters of the function being optimized, r_1 and r_2 are two independently generated random numbers in the range $[0, 1]$ and n represents the index of the best particle in the neighbourhood of a particle. The other vectors are defined as: $X_i = [X_{i1}, X_{i2} \dots X_{iD}] \equiv$ Position of i -th particle; $V_i = [V_{i1}, V_{i2} \dots V_{iD}] \equiv$ Velocity of the i -th particle; $B_i \equiv$ Best position of the i -th particle, and $B_i^n \equiv$ Best position found by the neighbourhood of the particle i . When the convergence criterion is satisfied, the best particle (with its position) found so far is taken as the solution to the problem.

2.2 AdaBoost for Object Detection

Viola and Jones' system [11] provides a typical approach to object detection. It uses a set of simple Haar-wavelet like features shown in Figure 1. A feature is calculated by subtracting the sum of pixel greyscale values of the white area of the rectangle from the dark area.

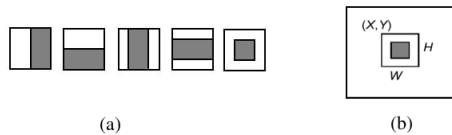


Fig. 1. (a) Standard Haar-like features. (b) The parameters of a feature. (X, Y) : position of the detection window, W and H : width and height of the feature.

The Adaboost algorithm iterates over T rounds. In each round, a new weak classifier $h_t()$ is constructed and added to the sequence of weak classifiers produced previously that ultimately together form the strong classifier. Through the run of the algorithm, the training examples are manipulated through *weighting*. The examples that are successfully classified by the previous weak classifier are given less weight than those misclassified in order to focus on hard examples, for instance those examples that are close to the decision boundary. The new weighted examples are provided again to the system to construct a new weak classifier in the subsequent round.

The pseudocode of the discrete Adaboost algorithm is listed in Algorithm 1. It is called discrete because the output of the classifier is either $+1$ for positive or -1 for negative example.

Algorithm 1. Pseudocode for AdaBoost

Given N examples $(x_1, y_1), \dots, (x_i, y_i), \dots, (x_N, y_N)$ where $y_i \in \{-1, 1\}$

Initialize $w_{1,i} = 1/2m, 1/2l$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.

for $t = 1, \dots, T$ **do**

(1) for each feature j , train a classifier $h_j()$

(2) Evaluate the error of the classifier $\epsilon_j = \sum_{i=1} \omega_{t,i} \cdot b_i$

(3) Choose a classifier $h_t()$ with lowest error ϵ_t

Update weights: $\omega_{t+1,i} = \omega_{t,i} \beta_t^{1-b_i}$

where $b_i = 0$ if $h_t(x_i) = y_i, b_i = 1$ otherwise

with $\beta_t = \epsilon_t / (1 - \epsilon_t)$

end for

Output strong classifier:

$$H(x) = \begin{cases} 1 & \text{if } \text{sign}(\sum_{t=1}^T \alpha_t h_t) \text{ is positive} \\ -1 & \text{otherwise} \end{cases} \quad (3)$$

with $\alpha_t = \log(1/\beta_t)$

The classifier with the minimum error $h_j()$ at round t is taken to be the winner among all available classifiers. The final *strong* classifier $H(x)$ at the end of T rounds is a linear combination of the weighted weak classifiers, where each classifier is weighted by a parameter α_t proportional to its error rate ϵ_t .

3 New PSO Based Approaches to Object Detection

3.1 AdaBoostPSO1 with Simple Weak Classifiers

We have developed two PSO methods for object detection. *AdaBoostPSO1* uses PSO to select good features only and the weak classifiers use a kind of decision stump.

Particle Encoding. A particle encodes the Haar-like feature parameters. The Haar-like features can be parameterized by the type (one of the types shown in

Type	X	Y	W	H

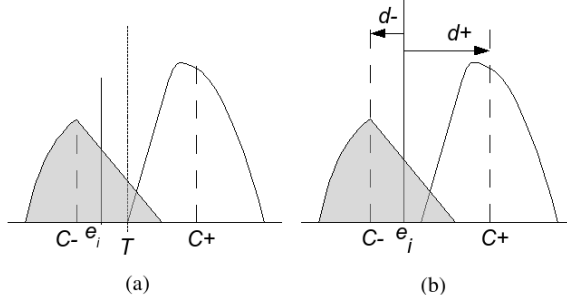
Fig. 2. Particle Encoding in *AdaBoostPSO1*

Fig. 3. (a) *AdaBoostPSO1*: using decision stump. C_- and C_+ are computed by taking the means of negative and positive examples respectively. Example e_i is labelled with class C_- as it is less than T . (b) *AdaBoostPSO2*: C_- and C_+ are found using PSO, and example e_i is labelled with class C_- as $d_- < d_+$.

Figure 1 (a)), upper-left position (X, Y) , with respect to the detection window, and the size (W, H) in Figure 1 (b). Figure 2 shows the encoding of the particle.

Weak Classifiers. Because the weak classifiers are only required to be better than random guessing, a decision stump, a depth-one decision tree which compares a single input feature to a threshold [13,14], has been widely used. The examples are assigned to the positive class if their single feature values are greater than the threshold and to the negative class otherwise. During learning in *AdaBoost*, each feature in the input feature vector is examined in turn to find the best feature. The threshold T is computed by taking the average of the means of the feature values from negative and positive examples as shown in Figure 3(a).

Fitness Function. We seek to use *AdaBoostPSO1* to find the best features to be used by the weak classifiers. The fitness function is to minimize the weighted error ϵ_j in *AdaBoost* (see Algorithm 1) for object detection/classification.

$$\epsilon_j = \sum_{i=1} \omega_{t,i} \cdot b_i \text{ where } b_i = 0 \text{ if } h_t(x_i) = y_i, b_i = 1 \text{ otherwise} \quad (4)$$

3.2 *AdaBoostPSO2* for Evolving Features and Weak Classifiers

AdaBoostPSO2 uses PSO for selecting good features and evolving corresponding weak classifiers in parallel. The particle encoding is shown in Figure 4.

In addition to the five parameters in *AdaBoostPSO1*, the particles in this PSO also include the centroids of the positive and negative examples. The C_- and C_+ are automatically evolved by PSO in this method rather than being simply calculated by taking the mean of the examples. Rather than selecting a feature and

Type	X	Y	W	H	C-	C+

Fig. 4. Particle Encoding in *AdaBoostPSO2*

then training a classifier to evaluate the goodness of the feature, the parameters of the feature and the associated weak classifier are considered as a whole to be optimized using PSO. Figure 3(b) shows how the weak classifier operates. For a given example e_i , the distance to the evolved centroids are computed and the example is labelled with the class that has the shortest distance. Because these two centroids are found in such a way to reduce the training error, the constructed weak classifier are expected to be more accurate. In this case, the optimum centroids are the two points that minimize the inner class distance and maximize the across class distance on the respective feature axis. In other words, two centroids (C_- and C_+) are found to maximize the following class separation criteria:

$$J(C_-, C_+) = \frac{|C_- - C_+|}{\frac{1}{m} \sum_{i=1}^m |C_- - e_i^-| + \frac{1}{l} \sum_{i=1}^l |C_+ - e_i^+|} \quad (5)$$

where m is the number of negative examples and l is the number of positive examples. Instead of maximizing this criteria based on the distances, it is maximized using PSO indirectly in such a way as to reduce the error rate. This is similar (in concept) to [15], where they maximize the class separation represented by Fisher discriminant formula through evolutionary minimizing of the classification error. However, that approach does not work for a single feature.

AdaBoostPSO2 uses the same fitness function as *AdaBoostPSO1*. *AdaBoostPSO2* is incorporated into AdaBoost to replace the exhaustive feature search and the weak classifier training. Thus, the three steps in the inner loop of Algorithm 1 are replaced by *AdaBoostPSO2*. In each round PSO is called to construct a new weak classifier.

Notice that the canonical particle swarm is used for optimization tasks where the weak classifier parameters to be optimized are *integer* numbers. Hence, an extra step is taken to round the encoded values to the nearest integer numbers before the particle is evaluated.

4 Experiment Design and Configurations

4.1 Data Set

The pasta image data set presented in the competition at GECCO 2006 [16] was used to evaluate the performance of the proposed approaches. Although the original problem in the competition [16] was for image segmentation [17] and the images were coloured, we used these images for object (pasta) detection in this paper and converted the colour images into gray-level images. The images are of size 1280×960 pixels that contain pasta pieces of different size, position and rotation with varied lighting conditions. The background of some images are



Fig. 5. The training images used to extract training examples. The first image shows the marking to create the examples (white areas are positive examples and black areas are negative examples).

noisy, including some pieces of alphabet soup, whose intensities are very similar to the pasta objects, making the detection problem even more challenging. Figure 5 shows some examples of these images used in the training process.

4.2 Object Detection Process

The object detection process using both PSO approaches consists of a training step and a testing step. In the training process, both AdaBoostPSO algorithms are applied to the training data set of object “cutout” examples. As the pasta pieces are of different sizes and rotations and the images are of large size, it is not so straightforward to exactly cut out the pastas from the large images. In this paper, we implement the following procedure to create training examples. On the training images, the pasta pieces are firstly marked differently from the background. A scanning window, of size 61×61 pixels, moves across the large training images in discrete jumps considering the current position as a pasta (positive) example if it stands over a marked “white” spot, and a position is considered a negative (non-pasta) example if the scanning window stands on a “black” marked spot. An example marked image is shown in the first column in Figure 5. In this way, a window could be still considered a positive example if it includes some part of a pasta and some pieces of background. Thus, the positive examples contain different parts of the pasta pieces rather than one complete piece. In addition, these parts could be quite different as the pasta pieces have different rotations, shapes and brightness. Following this procedure, 1297 positive examples and 2434 negative example were generated to form the training set.

The trained classifiers were then used as a template detector, in a moving window fashion, to detect the pasta objects over the *test set* of full images. The detection window size is the same as the size the training object cutout examples (61×61 pixels) and moves pixel by pixel over the large testing images. If a window is detected to be of pasta type, the centered pixel under the detection window is marked as an object (pasta) pixel. In this way, the pasta pixels are detected and an indication of the whole pastas (size and rotation) will be indicated.

4.3 Experiment Configuration

In all the experiments, both PSO approaches used the ring topology. Considering the computational effort in the training phase, the two PSO approaches used the same population size of 20 and maximum iteration of 100 for comparison purposes, so there are 2000 different features generated during the run

to evolve to the best feature and constructing a weak classifier. Note that this number is much smaller than that needed to be evaluated using the exhaustive search in Adaboost (roughly searching for 2×10^{11} features to find a single weak classifier), where a large percentage of them are redundant and do not have any discriminative power.

The particles are initialized within a range that does not exceed the size of the detection window and the two centroids parameters are initialized in the range $[-20000, 20000]$. This value is set after preliminary experimentation, and actually it is not critical as the distance between the centroids is of concern.

As it is too expensive to use the original Adaboost algorithm for the pasta detection process, we did not include it in our comparison. The next section will describe the experiment results of the two proposed AdaBoostPSO approaches.

5 Experimental Results

To measure the performance of the two proposed AdaBoostPSO approaches, we used the Receiver Operating Characteristic (ROC) curve for the test set. To compute “points” on the ROC curves, we used the large images in the test set to mark and generate 2941 positive examples and 3411 negative examples. The two marked test images are shown in Figure 6. The points on the ROC curves were obtained by evaluating the strong classifier against these examples and sliding the confidence threshold over the $(-\infty, +\infty)$ space. These were done by taking the average results of 100 independent runs. The ROC curves for the two approaches are shown in Figure 6. In this figure, the dashed curve represents the performance of the AdaboostPSO1 approach and the solid curve corresponds to the AdaboostPSO2 method.

As can be seen from Figure 6, both curves are much higher than the diagonal from (0, 0) to (1, 1), showing that both new PSO based approaches performed the pasta detection quite well. The solid curve is higher than the dashed curve at almost all false positive rates, showing that AdaboostPSO2 achieved better performance than AdaboostPSO1.

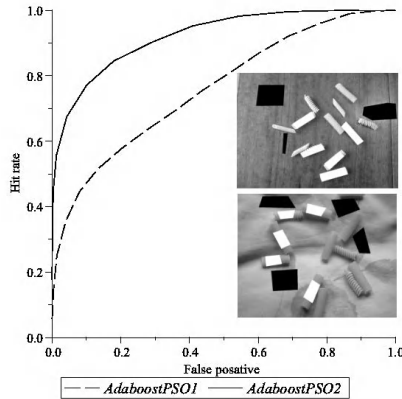


Fig. 6. ROC curves of the AdaboostPSO1 and Adaboost2 approaches

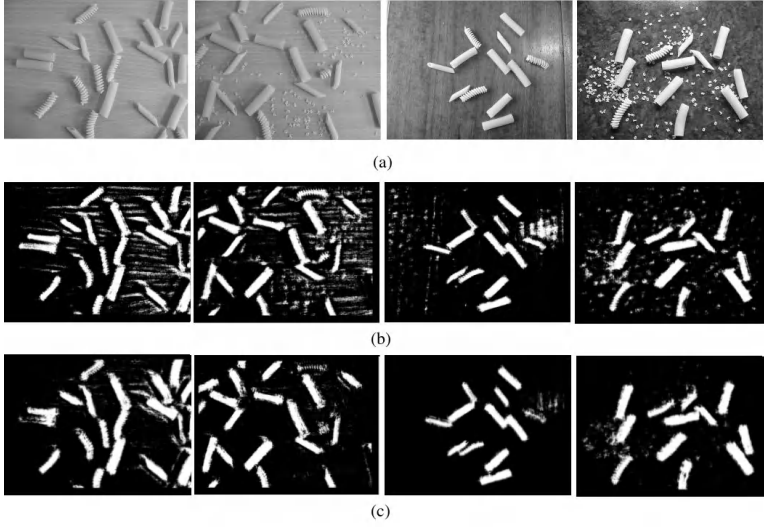


Fig. 7. (a) The testing images; (b) sweeping maps produced by AdaboostPSO1; (c) sweeping maps generated by AdaboostPSO2

To give an intuitive view of the detection results, Figure 7 shows the object sweep maps of pasta images in the test set. The images in the first row are the original images that were not used in the training process. The second row shows the corresponding sweeping maps achieved by AdaboostPSO1 and the third row show the sweeping maps of the original images achieved by AdaboostPSO2. In these sweeping maps, white pixels represent the “pasta pixels” detected by the two approaches. Clearly, these sweeping maps contain a number of false positives, that is, some non-pasta pixels (either noisy background or some alphabet soups) were incorrectly detected as pasta pixels. The sweeping maps confirm that both approaches successfully detected all pastas in the images, but AdaboostPSO2 produced a smaller number of false positives for all the examples images. In particular, AdaboostPSO2 is much more tolerant to the noisy background than AdaboostPSO1, and is also much better in discriminating pasta pixels from the alphabet soup. These results suggest that using PSO to select good features and evolve corresponding weak classifiers in AdaBoost is better than to select features only for the pasta detection problem examined in this paper.

6 Conclusions

The goal of this paper was to investigate a new approach using PSO within Adaboost for pasta detection. The goal was successfully achieved by using PSO for selecting good features and evolving weak classifiers in parallel within the Adaboost algorithm for object detection. In this way, the original time consuming exhaustive search in AdaBoost was successfully avoided.

Two PSO based methods were developed in this paper. *AdaboostPSO1* considers using PSO to evolve and select the good features only and the weak

classifiers use a kind of decision stump. *AdaBoostPSO2* considers using PSO for both selecting the good features and evolving weak classifiers in parallel. The experiment results show that both approaches performed quite well for the pasta detection problem, and that using PSO for selecting good individual features and evolving associated weak classifiers in AdaBoost is more effective than for selecting features only for this problem. We will investigate this approach on other object detection problems in the future.

References

1. Brown, G., Wyatt, J., Harris, R., Yao, X.: Diversity creation methods: A survey and categorisation. *Journal of Information Fusion* (Special issue on Diversity in Multiple Classifier Systems) 6, 5–20 (2005)
2. Valentini, G., Masulli, F.: Ensembles of learning machines. In: Plaice, J., Kropf, P.G., Schulthess, P., Slonim, J. (eds.) *DCW 2002*. LNCS, vol. 2468, pp. 3–20. Springer, Heidelberg (2002)
3. Krogh, A., Vedelsby, J.: Neural network ensembles, cross validation, and active learning. *Neural Information Processing Systems* 7, 231–238 (1995)
4. Opitz, D., Maclin, R.: Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research* 11, 169–198 (1999)
5. Ji, C., Ma, S.: Combinations of weak classifiers. *IEEE Transactions on Neural Networks* 8, 32–42 (1997)
6. Breiman, L.: Bagging predictors. *Machine Learning* 24, 123–140 (1996)
7. Valiant, L.: A theory of the learnable. *Communications of the ACM* 27(11), 1134–1142 (1984)
8. Kearns, M.J., Valiant, L.G.: Cryptographic limitations on learning boolean formulae and finite automata. *Journal of ACM* 1, 67–95 (1994)
9. Kearns, M.J., Valiant, L.G.: The boosting approach to machine learning: An overview. In: *Nonlinear Estimation and Classification*. Springer, Heidelberg (2003)
10. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences* 55(1), 119–139 (1997)
11. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, pp. 511–518 (December 2001)
12. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: *Neural Networks*, pp. 1942–1948 (1995)
13. Barger, D., Viola, P., Simard, P.: Boosting-based transductive learning for text detection. In: *Eighth International Conference on Document Analysis and Recognition*, vol. 2, pp. 1166–1171 (2005)
14. Bradski, G., Kaehler, A., Pisarevsky, V.: Learning-based computer vision with intel's open source computer vision library. *Intel Technology Journal* 09(1), 119–130 (2005)
15. Sierra, A., Echeverria, A.: Evolutionary discriminant analysis. *IEEE Transactions on Evolutionary Computation* 10(1), 81–92 (2006)
16. Pasta segmentation competition,
<http://cswww.essex.ac.uk/staff/rpoli/GECC02006/pasta.htm>
17. Cagnoni, S., Mordonini, M., Sartori, J.: Particle swarm optimization for object detection and segmentation. In: *Giacobini, M. (ed.) EvoWorkshops 2007*. LNCS, vol. 4448, pp. 241–250. Springer, Heidelberg (2007)

Adaptive Non-uniform Distribution of Quantum Particles in mQSO

Krzysztof Trojanowski

Institute of Computer Science, Polish Academy of Sciences
Ordona 21, 01-237 Warsaw, Poland
trojanow@ipipan.waw.pl

Abstract. This paper studies properties of quantum particles rules of movement in particle swarm optimization (PSO) for non-stationary optimization tasks. A multi-swarm approach based on two types of particles: neutral and quantum ones is a framework of the experimental research. A new method of generation of new location candidates for quantum particles is proposed. Then a set of experiments is performed where this method is verified. The test-cases represent different situations which can occur in the search process concerning different numbers of moving peaks respectively to the number of sub-swarms. To obtain the requested circumstances in the testing environment the number of sub-swarms is fixed. The results show high efficiency and robustness of the proposed method in all of the tested variants.

1 Introduction

Recent research of non-stationary optimization with particle swarm approach showed its high efficiency. Especially multi-swarm with mixed types of particles proved its usefulness and outperformed other metaheuristics. Approaches with static and varying number of sub-swarms [1], [2], [3] as well as approaches with adaptive number of species in the swarm [4], [5], [6] have been researched. The results of the research showed a key role of appropriate sub-swarms management during the process of search. In this paper multi-swarm approach is also a subject of interest, however, the main goal is studying properties of the particles rules of movement. In the approach presented in this paper the number of sub-swarms is constant during the process of search as in the approaches developed by Blackwell and Branke in [3]. The multi-swarms presented in [3] consist of two types of particles: neutral ones and quantum or charged ones. In most cases the multi-swarms with quantum particles outperformed those with charged ones and therefore in this paper just the multi-swarm with quantum and neutral particles called mQSO is studied.

Quantum particles described in [3] and neutral ones obey different rules of movement. Quantum particle candidates for a new location are uniformly distributed within a so called quantum cloud i.e. a hyper-sphere surrounding the original location of the particle. All the locations in the hyper-sphere have the same probability to be selected. It is necessary to mention here that such rules

of movement assume that the search space is defined in $\mathbb{R}^n$ where euclidean distances can be calculated. Besides when we use quantum particles an additional assumption about the transformation rules of the set of real-world possible solutions into the search space is needed. Since the quantum rules of movement treat equally each of the dimensions of the search space the transformation should build a scaled space. The scaling factors for the dimensions should satisfy the expectation that the same movement along each of them has the same strength of change when transformed back to the set of the real-world possible solutions. Besides the feasible ranges of the real-world parameters should be converted in the way which turns the feasible part of the space into a hyper-cube i.e. with the same side-length for each of dimensions. This assumption does not concern neutral particles where movement along each of the dimensions is calculated separately but it has to be taken into account for quantum ones. An appropriate transformation of the domain of the real-world optimization problem into the domain of the optimized model allows to avoid this possible lack of symmetry in strength of changes and in the ranges of the space.

Application of quantum particles proved to be efficient especially in the case when the number of moving peaks is comparable to the number of sub-swarms. However, in this paper we study the role of rules of movement for two different types of the optimization situation which require different response of the optimizer. The first one is the case when a global optimum hill actually occupied by the particles moves, however, it still remains the global optimum and some of the particles remain in its basin of attraction after the change. The second one appears when after the change the occupied hill stops to be the global optimum and none of the hills currently controlled by the particles takes over the role of the global optimum. In the latter case the immediate extensive exploration of the search space is needed.

To cope efficiently with both situations a new method of generation of the new location candidates is proposed and experimentally verified. The method is a hybrid approach where quantum particle rules and the rules based on α -stable distribution are joined together. Obtained results show that the proposed method outperforms the method based on the idea of the quantum cloud. The paper consists of seven sections. Section 2 describes mQSO approach while Sec. 3 – rules of the new location candidates generation for the proposed method. In Sec. 4 the selected benchmark is briefly presented. Sections 5 and 6 include the plan of experiments and the results. Section 7 concludes the paper.

2 Quantum Multi-swarm

A PSO optimizer is equipped with a set of particles $\mathbf{x}_i$ where $i \in [1, \dots, N]$. Each of the particles represents a solution in an n -dimensional real valued search space. For the search space a fitness function $f(\cdot)$ is defined which is used to evaluate the quality of the solutions. Each of the particles is equipped with so called particle attractor $\mathbf{y}_i$ which stores the best solution found by the i -th particle. Besides the swarm has its own attractor $\mathbf{y}^*$ with the best solution found by the swarm. The

algorithm is based on an iterated process of continuous movement of the particles in the search space. Every iteration starts with update of attractors and then two main actions are executed: first the velocity of each of the particles is calculated and then all the particles change their location in the search space according to the new values of velocity vectors and the kinematic laws. Formally for every iteration t of the search process every j -th coordinate of the velocity vector $\mathbf{v}$ as well as the coordinate of the location $\mathbf{x}$ undergo the following transformation [7]:

$$\begin{aligned} v_j^{t+1} &= \chi(v_j^t + c_1 r_1^t (y_j^t - x_j^t) + c_2 r_2^t (y_j^{*t} - x_j^t)) , \\ x_j^{t+1} &= x_j^t + v_j^{t+1} , \end{aligned} \quad (1)$$

where χ is a constriction factor and $\chi < 1$ and c_1 and c_2 control the attraction to the best found personal and global solutions, respectively.

The basic idea presented above has been extended for non-stationary optimization applications. One of the significant changes in this scheme is an introduction of multi-swarm. Each of the sub-swarms is treated as an independent self-governing population whose members are not influenced by any of the neighbors. However, there exist external mechanisms, which periodically perform some actions based on the information about the state of search of the entire swarm [3].

To guarantee the appropriate distribution of the sub-swarms over the entire search space an *exclusion* mechanism exists, which eliminates sub-swarms being located too close to each other. When the sub-swarms' attractors are closer than the euclidean distance r_{excl} , the occupation of the same optimum is most likely to occur. In this case one of the sub-swarms is selected to be eliminated and a new one is generated from scratch. Another mechanism of sub-swarms' management called *anti-convergence* helps to detect new peaks in the landscape. Whenever all the sub-swarms converge i.e. when for each of them all the particles are located in an area limited by the radius r_{conv} , the sub-swarm with the worst attractor $\mathbf{y}^*$ is randomized. In the base version of mQSO the latter mechanism is off.

The last of the sub-swarms management mechanisms described in [3] is based on mixing of types of particles in sub-swarms. In the presented research the mixed sub-swarms consist of two types of particles. While the location of the neutral particles is evaluated according to formula (1), the quantum ones change their location according to the analogy with quantum dynamics of the particles. Idea of quantum particle proposed in [2] originates from the quantum model of atom where the trajectories of electrons are described as quantum clouds. Adaptation of this idea to the model of movement of the particles rejects kinematics laws used in classic PSO for evaluation of a distance traveled by the particle with a constant velocity in a period of time. Instead of this the new position of the quantum particle is randomly generated inside a cloud of the limited range surrounding $\mathbf{y}^*$ i.e. the current sub-swarm attractor. In quantum model the particle's speed becomes irrelevant, because every location inside the cloud can be chosen as a new location of the particle with a non-zero probability. This idea of particles' movement is similar to the one called *bare-bones* PSO [8] however some differences can be observed. In bare-bones PSO all the particles were moved

randomly while in [3] there exist two types of particles in the swarm: quantum and neutral ones. In the former approach the Gaussian distribution is in use where the mean is in the half way between the particle and the swarm attractor and scale parameter equals the euclidean distance between the two. For quantum particles the center of the quantum cloud is in the swarm attractor and the range of the cloud is limited by a constant parameter.

Finally it is important to stress that in the presented version of the algorithm there is no strategy of detecting the appearance of change in the fitness landscape. Since our main goal was studying the properties of the different distributions of the quantum particles, we assumed that it would just introduce yet another unnecessary bias into the obtained values of offline error and make their analysis even more difficult. Therefore information about the appearance of a change is known to the system instantly as it happens however the system does not know what kind of change this is. So when the change appears, the first reaction of the system is to refresh information about the quality of solutions maintained currently. All the solutions stored in both neutral and quantum particles are reevaluated and the attractors have to be reelected. Attractors of neutral particles are overwritten by their current solutions and sub-swarms' attractors are overwritten by the best solutions in the sub-swarms.

3 Movement of Quantum Particles

The new distribution method considered in this paper is based on a two-phase mechanism which is similar to the two-phase mechanism described in [9]. In the first phase a new point uniformly distributed within the hyper-sphere of the range $\delta = r \cdot (D_w/2)$ is found, where r is a control parameter and D_w is a width of the feasible part of the domain, i.e. a distance between a lower and an upper boundary of the search space. The difference between the presented approach and the one from [9] is in the rules of the new point generation in the first phase. Here the new point is found within a hyper-sphere while in [9] the point is found on the surface of the hyper-sphere. One can observe that this first phase works in exactly the same way as the rules of the new location generation in the quantum cloud of limited range presented in [2] since all the locations within the hyper-sphere have also the same probability.

The second phase is the same as in [9]. The generated point is moved forward according to the direction defined by the original location and the point. Eventually the point is located in the new distance d' from the original position which is calculated as follows:

$$d' = d \cdot \exp(-f'(\mathbf{x}_i)) \cdot S\alpha S(0, 1), \quad (2)$$

where d is a distance from the original position obtained in the first phase, $f'(\mathbf{x}_i)$ is the fitness of the i -th solution $\mathbf{x}_i$ normalized in $[0, 1]$ respectively to the fitness values of all the solutions in the swarm:

$$f'(\mathbf{x}_i) = \frac{f(\mathbf{x}_i) - f_{\min}}{(f_{\max} - f_{\min})}, \quad (3)$$

$$f_{\max} = \max_{j=1,\dots,N} f(\mathbf{x}_j) \text{ and } f_{\min} = \min_{j=1,\dots,N} f(\mathbf{x}_j)$$

and $S_{\alpha}S(0,1)$ represents α -stable symmetric distribution variate. This is an isotropic distribution where the locations are distributed equally in all directions i.e. none of directions is distinctive in any sense. Application of two-phase mechanism with $S_{\alpha}S(0,1)$ outperforms the approach with uniform distribution inside a quantum cloud because allows to control the density of distribution of new location candidates in different distances from the location of the original. For small values of α a large number of the candidates is assembled close to the original albeit there is also a number of candidates located in further distance (so called *heavy-tailed distribution*). For high values of α almost all of the candidates are spread over the wider range from the original while the candidates located far from the original appear seldom if ever.

The α -stable distribution is controlled by four parameters: stability index α ($\alpha \in (0 < \alpha \leq 2)$), skewness parameter β , scale parameter σ and location parameter μ . In the practical implementation the Chambers-Mallows-Stuck method of generation of the α -stable symmetric random variables [10] can be used. The method for $\sigma = 1$ and $\mu = 0$ is presented in the equation (4). To calculate the α -stable distributed random variate X two another independent random variates are needed: U , which is uniformly distributed on $[-\pi/2, \pi/2]$ and an exponential random variate W obtained with rate parameter $\lambda = 1$:

$$X = \begin{cases} S_{\alpha,\beta} \cdot \frac{\sin \alpha(U+B_{\alpha,\beta})}{(\cos U)^{1/\alpha}} \cdot \left(\frac{\cos(U-\alpha(U+B_{\alpha,\beta}))}{W} \right)^{(1-\alpha)/\alpha}, & \text{iff } \alpha \neq 1, \\ \frac{2}{\pi} \left(\left(\frac{\pi}{2} + \beta U \right) \tan U - \beta \ln \frac{\frac{\pi}{2} W \cos U}{\frac{\pi}{2} + \beta U} \right), & \text{iff } \alpha = 1. \end{cases} \quad (4)$$

where $B_{\alpha,\beta} = \alpha^{-1} \arctan(\beta \tan \frac{\pi\alpha}{2})$, and $S_{\alpha,\beta} = \left(1 + \beta^2 \tan^2 \frac{\pi\alpha}{2}\right)^{1/(2\alpha)}$.

In the presented research the symmetric version (called $S_{\alpha}S(\mu, \sigma)$, i.e. symmetric α -stable distribution) is applied i.e. β is set to 0. For $\alpha = 2$ the $S_{\alpha}S(\mu, \sigma)$ reduces to the Gaussian $N(\mu, \sigma)$ and for $\alpha = 1$ the Cauchy $C(\mu, \sigma)$ is obtained.

Since the main focus in this paper is not about the constrained optimization, we selected a very simple procedure of immediate repairing unfeasible particles. Clearly the j -th coordinate of the solution $\mathbf{x}$ breaking its box constraints called *lo* (lower) and *hi* (upper) is trimmed to the exceeded limit, i.e.:

$$\text{if } x_j < lo_j \text{ then } x_j = lo_j, \quad \text{if } x_j > hi_j \text{ then } x_j = hi_j. \quad (5)$$

The procedure is applied in the same way to both types of the particles, the neutral and the quantum ones. In case of neutral particles the velocity vector $\mathbf{v}$ of the repaired particle remains unchanged even if it still leads the particle outside the acceptable search space.

4 Applied Measure and the Benchmark

In the performed experiments we used the *offline error* (briefly *oe*) measure [11,12]. During the process of searching *oe* starts to be evaluated only after some number of changes in the fitness landscape. The initial search process was not used to measure performance since this represents the settling period of the swarm with a high value of error and is not representative of the behavior. For compatibility with experiments published by others the number of evaluations between subsequent changes of the fitness landscape equals 5000. During a single experiment the fitness landscape changes 110 times (however for the first 10 changes *oe* is not evaluated). Every experiment was repeated 50 times.

As a dynamic test-bed a MPB [11] generator was selected which is publicly available in the web [13]. In MPB we optimize in a real-valued n -dimensional search space and the fitness landscape is built of a set of unimodal functions. The functions are individually controlled by the parameters allowing to create different types of changes. The selected benchmark MPB allows control over the intensity of appearance of both types of optimization situations mentioned in Introduction by controlling the number of peaks. Since the number of sub-swarms is constant in our experiments and is set to 10 the first situation appears for 10 moving peaks in the landscape. In this case it is expected that every peak is controlled by another sub-swarm and all we need is to make the sub-swarms able to follow their hills. The second type of changes occurs when the number of moving peaks is much higher than the number of sub-swarms, for example when there are 50 moving peaks and 10 sub-swarms. Every change in the landscape causes not only a move of the peaks but also changes of their heights so it hardly even happens that the highest peak remains the highest one after the change.

The remaining parameters of MPB were set exactly the same as specified in scenario 2 of this benchmark. The fitness landscape was defined for $\mathbb{R}^5$ with boundaries for each of dimensions set to $[0; 100]$. For the search space there exist a set of moving peaks which vary their height randomly within the interval $[30; 70]$, width within $[1; 12]$ and coordinates of the top by 1.

5 Plan of Experiments

The algorithm parameters' settings applied to the experiments presented below originate from [3]. For MPB sc. 2 with 10 moving peaks the best results are obtained where there are 10 sub-swarms and each of them consists of five neutral particles and five quantum ones (see Table III in [3]). The total population of particles consists of 100 solutions divided equally into 10 sub-swarms. The values of pure PSO parameters are: $c_{1,2} = 2.05$ and $\chi = 0.7298$. QSO parameters: $r_{\text{excl}} = 31.5$ and $r_{\text{conv}} = 30$.

There are four cases to compare: two versions of the algorithm (without and with anti-convergence) and two versions of the testing environment (with low and with high number of moving peaks). A set of tests was performed for each of the cases. Each set was based on variation of values of two parameters of the

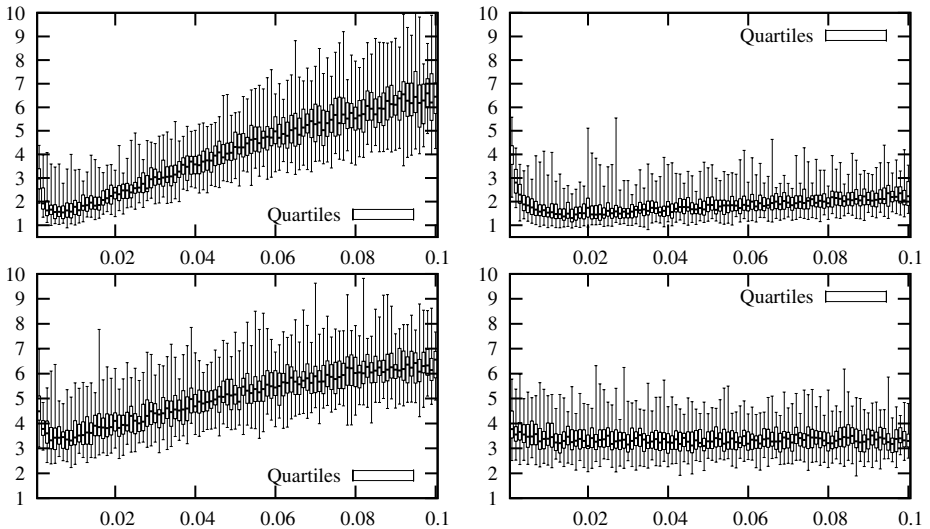
Table 1. The best obtained mean value of oe for mQSO with a uniform distribution of new locations inside a quantum cloud and for the proposed new type of distribution

No. of peaks; mQSO version	quantum cloud			α -stable dist.			
	oe	std.dev.	r	oe	std.dev.	r	α
10 peaks; base	1.6133	0.3891	0.007	1.4293	0.3710	0.016	1.65
50 peaks; base	3.4093	0.5129	0.007	3.1321	0.4716	0.031	0.80
10 peaks; anti-conv.	3.6800	0.7727	0.013	3.0522	0.7515	0.029	1.10
50 peaks; anti-conv.	3.3567	0.4056	0.013	2.9510	0.3647	0.031	1.20

rules of movement: α and r . The former parameter varied from 0.05 to 2 with step 0.05 while the latter – from 0.001 to 0.15 with step 0.001. It gave 6000 configurations of the two parameters per set and allowed to build reliable graphs with characteristics of the search engines.

6 Results of Experiments

In the first stage of the research some of the experiments with the base version of the mQSO algorithm and a uniform distribution of new locations inside a quantum cloud presented in [3] were repeated to make them comparable to the results presented in this paper. It was necessary since in our case a period of the

**Fig. 1.** Box-and-whisker diagrams of oe for base version of mQSO with the quantum cloud distribution (the left hand side) and with the proposed method of quantum particles movement (the right hand side) for the case with 10 (top row) and 50 (bottom row) moving peaks

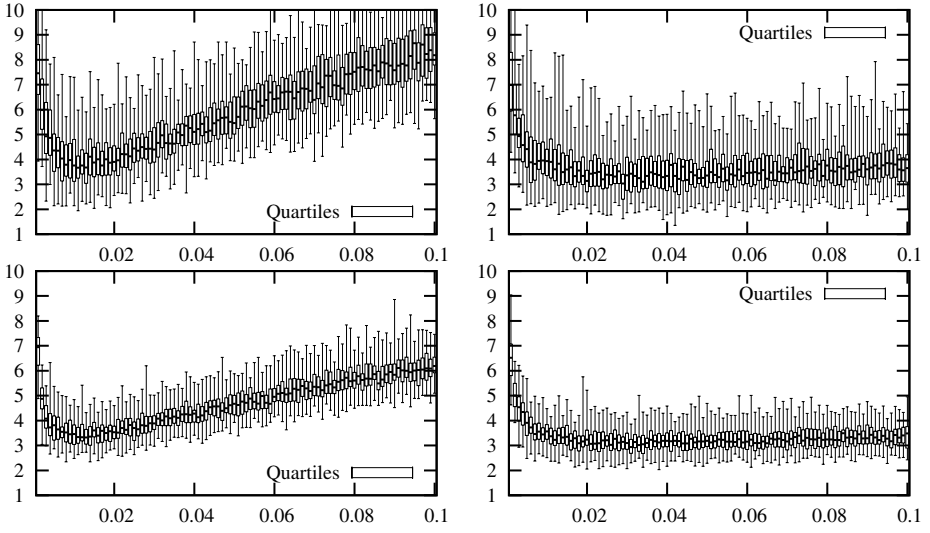


Fig. 2. Box-and-whisker diagrams of oe for mQSO with anti-convergence with the quantum cloud distribution (the left hand side) and with the proposed method of quantum particles movement (the right hand side) for the case with 10 (top row) and 50 (bottom row) moving peaks

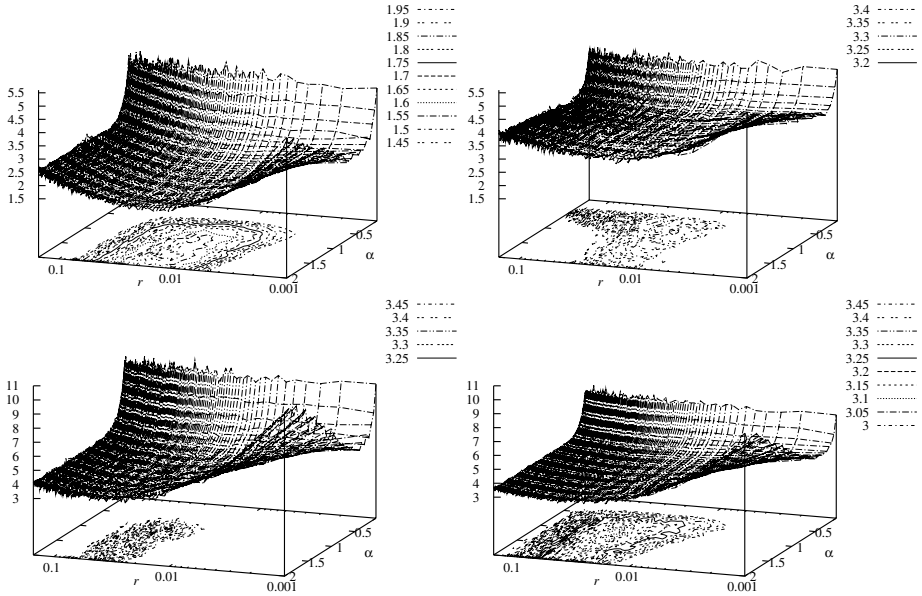


Fig. 3. Characteristics of oe for the base version of mQSO (top graphs) and for mQSO with anti-convergence (bottom graphs) for the case with 10 moving peaks (graphs on the left) and with 50 moving peaks (graphs on the right)

first 10 changes in the environment is excluded from calculating the offline error which makes the values of oe significantly smaller than those in [3]. We repeated experiments for a series of values of r parameter which varied from 0.001 to 0.1 with step 0.001. The best mean value of oe are presented in Table 1.

Figures 1 and 2 present graphs with box-and-whisker diagrams of oe including five quartiles for both types of testing environment: with 10 and with 50 moving peaks. They were obtained for both version of mQSO – without and with anti-convergence – and for both types of quantum particle distribution. In case of the quantum cloud approach the tests were performed for the range of r from 0.001 to 0.1 with step 0.001. In the other cases there are two parameters to tune: r and α . The α parameter was fixed to the value giving the best results (see Table 1) while the r parameter varied in the same range as for quantum cloud. Characteristics of the search engines equipped with the new type of quantum particle distribution are presented in Fig. 3. They show obtained median values where a boxcar smoothing (with a window of size 3) has been applied.

7 Conclusions

In this paper a new method of generation of the new location candidates for quantum particles is presented. At first view, when we look at Table 1 the approach with quantum cloud is outperformed by the proposed new one. The difference is confirmed by the significance levels obtained with Student's t-tests. For all rows in Table 1 the significance level is lower than 0.05 and equals respectively: $p = 0.01851$, $p = 0.00641$, $p = 0.00009$ and $p = 0.000001$.

What is more important, graphs with box-and-whisker diagrams of offline error (Fig. 1 and 2) show that the proposed new method is much more robust i.e. resistant to imprecise tuning of its parameters. Clearly, even in case of too big value of r parameter the returned value of offline error should not increase significantly in opposite to the quantum cloud method where tuning of the r parameter is necessary.

It is interesting how the characteristics with smoothed median values (Fig. 3) for the case with 10 moving peaks differ from the ones for 50 moving peaks. Except for the difference of the overall level of values of offline error between these graphs which is rather obvious there is also another difference of the quality of results obtained for values of α less than 1. In case of 10 moving peaks the base version of mQSO returned the best results for α being greater than 1. In case of 50 moving peaks they were obtained for α in the range between 0.4 and 1. But when mQSO is equipped with anti-convergence the best results for both test-cases are returned for similar values of α being close to 1 (Table 1).

The anti-convergence is regarded as a remedy in case where the number of moving peaks is higher than the number of sub-swarms. Our experiments confirm this and show also that this mechanism quite significantly decreases the results when the two numbers are the same. Compared with the anti-convergence approach the new distribution method performs well for wider range of fitness landscapes. Besides when we chose the base version of mQSO we benefit more

when we add the new distribution method than the anti-convergence. However, it is necessary to stress that for 50 moving peaks the highest quality was obtained for the approach including both extensions (see Table 1 – last row, and also Fig. 3 – bottom-right graph). It means that in that test-case they can reinforce each other when used together.

Acknowledgments

The research reported in this paper was supported by the grant N 516 384734 from the Polish Ministry of Science and Higher Education.

References

1. Blackwell, T.: Particle swarm optimization in dynamic environments. In: Yang, S., Ong, Y.S., Jin, Y. (eds.) *Evolutionary Computation in Dynamic and Uncertain Environments*. Studies in Computational Intelligence, vol. 51, pp. 29–49. Springer, Heidelberg (2007)
2. Blackwell, T., Branke, J.: Multi-swarm optimization in dynamic environments. In: Raidl, G.R., Cagnoni, S., Branke, J., Corne, D.W., Drechsler, R., Jin, Y., Johnson, C.G., Machado, P., Marchiori, E., Rothlauf, F., Smith, G.D., Squillero, G. (eds.) *EvoWorkshops 2004*. LNCS, vol. 3005, pp. 489–500. Springer, Heidelberg (2004)
3. Blackwell, T., Branke, J.: Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Tr. Evolutionary Computation* 10(4), 459–472 (2006)
4. Li, X.: Adaptively choosing neighborhood bests in a particle swarm optimizer for multimodal function optimization. In: Deb, K., et al. (eds.) *GECCO 2004*. LNCS, vol. 3102, pp. 105–116. Springer, Heidelberg (2004)
5. Li, X., Branke, J., Blackwell, T.: Particle swarm with speciation and adaptation in a dynamic environment. In: *GECCO 2006: Proc. Conf. on Genetic and Evolutionary Computation*, pp. 51–58. ACM Press, New York (2006)
6. Parrot, D., Li, X.: Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *IEEE Trans. Evol. Comput.* 10(4), 440–458 (2006)
7. Clerc, M., Kennedy, J.: The particle swarm-explosion, stability, and convergence in a multi-dimensional complex space. *IEEE Tr. Evolutionary Computation* 6(1), 58–73 (2002)
8. Kennedy, J.: Bare bones particle swarms. In: *Proc. of the IEEE Swarm Intelligence Symposium 2003 (SIS 2003)*, pp. 80–87. IEEE Press, Los Alamitos (2003)
9. Trojanowski, K.: Non-uniform distributions of quantum particles in multi-swarm optimization for dynamic tasks. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) *ICCS 2008, Part I*. LNCS, vol. 5101, pp. 843–852. Springer, Heidelberg (2008)
10. Chambers, J.M., Mallows, C.L., Stuck, B.W.: A method for simulating stable random variables. *J. Amer. Statist. Assoc.* 71(354), 340–344 (1976)
11. Branke, J.: Memory enhanced evolutionary algorithm for changing optimization problems. In: *Proc. of the Congress on Evolutionary Computation*, vol. 3, pp. 1875–1882. IEEE Press, Piscataway (1999)
12. Branke, J.: *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers, Dordrecht (2002)
13. Branke, J.: The moving peaks benchmark,
<http://www.aifb.uni-karlsruhe.de/~jbr/MovPeaks/movpeaks/>

Genetically Evolved Fuzzy Rule-Based Classifiers and Application to Automotive Classification

Teck Wee Chua and Woei Wan Tan

Department of Electrical and Computer Engineering,
National University of Singapore,
4, Engineering Drive 3, Singapore 117576
{cteckwee, wwtan}@nus.edu.sg

Abstract. Type-2 fuzzy logic systems (FLSs) have been treated as a magic black box which can better handle uncertainties due to the footprint of uncertainty (FOU). Although the results in control applications are promising, the advantages of type-2 framework in fuzzy pattern classification is still unclear due to different forms of outputs produced by both systems. This paper aims at investigating if type-2 fuzzy classifier can deliver a better performance when there exists imprecise decision boundary caused by improper feature extraction method. Genetic Algorithm (GA) is used to tune the fuzzy classifiers under Pittsburgh scheme. The proposed fuzzy classifiers have been successfully applied to an automotive application whereby the classifier needs to detect the presence of human in a vehicle. Results reveal that type-2 classifier has the edge over type-1 classifier when the decision boundaries are imprecise and the fuzzy classifier itself has not enough degrees of freedom to construct a suitable boundary. Conversely, when decision boundaries are clear, the advantage of type-2 framework may not be significant anymore. In any case, the performance of a type-2 fuzzy classifier is at least comparable with a type-1 fuzzy classifier. When dealing with real world classification problem where the uncertainty is usually difficult to be estimated, type-2 fuzzy classifier can be a more rational choice.

Keywords: Type-2 Fuzzy, Fuzzy Rule-Based Classifier, Genetic-Fuzzy.

1 Introduction

The idea of incorporating type-2 fuzzy sets into a FLS framework stems from the need to model uncertainty in the description of antecedents and consequents in the system rule base. The type-2 FLS had the problem that it was envisaged as a computability expensive system due to the computational overhead associated with type-reduction and the use of the iterative Karnik-Mendel procedure [1]. Therefore, an important question arises: is it worth making use of type-2 FLS instead of type-1 FLS at the cost of complexity? Starczewski [2] shows that under certain conditions the output of both type-2 and type-1 FLSs are equivalent which might invalidate the type-2 approach in the majority of real application tasks. However, in most other circumstances the output differences

between both FLSs still exist. The subtle output differences might be critical in some applications such as precision control engineering. Type-2 FLSs have been applied successfully in control engineering field where type-2 fuzzy logic controllers (FLCs) are known to deliver better performances in face of uncertainties and imprecisions.

Although a considerable amount of effort has gone into type-2 fuzzy controller in the past, less research has been performed on the application of type-2 fuzzy in pattern classification. In [3], type-2 FLCs with non-hierarchical and hierarchical architecture was applied to the classification of battlefield ground vehicles. The input to the system is a set of acoustic features. The input is inherently noisy due to the variation of the vehicle traveling speed, along with the environmental variations (e.g., wind and terrain). To further model the input uncertainties, the input is modeled as an interval type-2 fuzzy set whose membership function (MF) is a Gaussian function that is centered at the measured value but with an uncertain standard deviation. Given the noisy acoustic inputs, it was observed that interval type-2 fuzzy rule-based classifier (FRBC) only gives marginally improvements over type-1 FRBC. The authors have raised a few important questions with regards to fuzzy pattern classification. These include in what way are type-2 FRBCs considered outperform T1 counterpart (e.g., in terms of the classification error rate, or generalizability or robustness), and how much uncertainty must be present in a problem so that it is worthy of trading the complexity of type-2 FRBC for better performance.

The performance of a classifier is heavily related to questionable choices that the designer of the classifier makes based in his/her insights into that problem [3]. One of them is the feature extraction method. An effective set of features can ease the design of classifier tremendously, especially for fuzzy classifier whereby the number of rules will increase exponentially with the increase of feature dimensions [4]. In contrast, if the feature selection is not optimum then the classification performance will be degraded. Unfortunately, it is difficult, and generally an open problem, to select an optimum set of features for different applications. Designer with experience may incorporate his/her knowledge about the classification problem. For example, by knowing that the ECG amplitude is a better feature to differentiate between ventricular tachycardia and ventricular fibrillation the designer can use this feature to improve the classifier performance [5]. On the other hand, designer with statistical background may try to use statistical tool such as Principal Component Analysis (PCA) or Linear Discriminant Analysis (LDA) to select a compact set of projected features. Therefore, one source of uncertainty in pattern classification is the ambiguity in feature selection. The objective of this paper is to investigate if type-2 FRBC can better handle the uncertainty associated with feature selection. The motivation is to investigate if the extra degrees of freedom provided by FOU may enable a type-2 FRBC to outperform type-1 counterpart. In the worst case scenario, if all FOUs of a type-2 FRBC disappear, then type-2 FRBC is immediately reduced to type-1 FRBC and there is no difference between the final outputs from both classifiers. This paper is organized as follows. Section 2 outlines the interval type-2 fuzzy

rule-based classifier (FRBC) and section 3 explains the training procedure with GA. The automobile classification problem and feature extraction method are explained in section 4 and sub-section 4.1 respectively. The experimental results are presented in sub-section 4.2 and finally section 5 offers concluding remarks.

2 Interval Type-2 Fuzzy Rule-Based Classifiers

This section introduces the interval type-2 FRBC. Fig. 1 shows the general structure of the proposed type-2 fuzzy rule-based classifier. There are six components in the architecture. The rule-base consists of M rules where each rule relates the domain $X_1 \times \cdots \times X_p \subseteq R^p$ to the range $Y \in R$ and can be expressed as the following intuitive IF-THEN statement:

$$R^j: \text{IF } x_1 \text{ is } \tilde{A}_1^j \text{ and } \cdots x_p \text{ is } \tilde{A}_p^j, \text{ THEN } y \text{ is } C^j$$

where R^j denotes the j th rule, $\tilde{A}_k^j$ is an interval type-2 antecedent set associated with the k th input variable x_k ($k = 1, \dots, p$), and C^j represents the consequent set associated with the output variable y . The role of the fuzzifier in a fuzzy system is to map each of the element, x'_k , in the input vector $\mathbf{x}' = (x'_1, \dots, x'_p)^T$ into the fuzzy set $\tilde{X}'_k$. This process provides a natural framework for handling uncertain input information. There is a variety of methods for performing fuzzification. The most common approach is singleton fuzzification, which maps a crisp input into the following MF:

$$\mu_{\tilde{X}'}(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} = \mathbf{x}' \\ 0 & \mathbf{x} \neq \mathbf{x}' \end{cases}$$

for $\forall \mathbf{x} \in \mathbf{X}$. Next, the inference engine component computes the firing strengths for each rule which expresses how well the fuzzified input $\tilde{X}'$ match the antecedents $\tilde{A}'$. For type-2 FRBC, the inference engine produces two firing strengths for each rule, the lower and upper firing strengths of the j th rule, $\underline{f}^j(\mathbf{x}')$ and $\bar{f}^j(\mathbf{x}')$, are computed as:

$$\underline{f}^j(\mathbf{x}') = \prod_{k=1}^p \sup_{x_k} [\mu_{\tilde{X}'_k}(x_k), \underline{\mu}_{\tilde{A}_k^j}(x_k)] \quad (1)$$

$$\bar{f}^j(\mathbf{x}') = \prod_{k=1}^p \sup_{x_k} [\mu_{\tilde{X}'_k}(x_k), \bar{\mu}_{\tilde{A}_k^j}(x_k)] \quad (2)$$

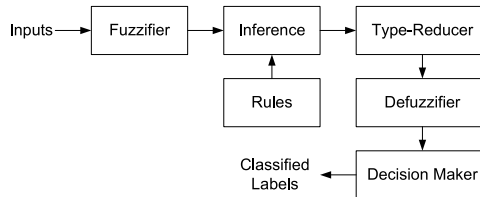


Fig. 1. Structure of type-2 classifier

where $\sup[\cdot]$ denotes supremum operation [1]. Before the final crisp output can be obtained, the output of the inference engine and the consequent must be processed. In a more general case where the consequent fuzzy sets $\tilde{C}^j$ are interval type-2 sets, the type-reduced set Y_{cos} can be computed with center-of-sets type reduction:

$$Y_{cos} = [y_l, y_r] = \int_{y^1 \in [y_l^1, y_r^1]} \cdots \int_{y^M \in [y_l^M, y_r^M]} \int_{f^1 \in [\underline{f}^1, \bar{f}^1]} \cdots \int_{f^M \in [\underline{f}^M, \bar{f}^M]} 1 \left/ \frac{\sum_{j=1}^M f^j y^j}{\sum_{j=1}^M f^j} \right. \quad (3)$$

where $[y_l^j, y_r^j]$ denotes to the centroid of the set $\tilde{C}^j$, which can be obtained from various methods defined in [1]. However, the consequent fuzzy sets in our classification problem correspond to the class labels and are represented by crisp number (singleton), the center-of-sets type-reduction above is simplified to height type-reduction by simply setting $y_l^j = y_r^j$. The type-reduced set which is an interval output, $[y_l(\mathbf{x}'), y_r(\mathbf{x}')] can be obtained via Karnik-Mendel iterative algorithm [6]. The type-reduced set is then defuzzified to the crisp output, y by simply taking the average of y_l and y_r , i.e.:$

$$y(\mathbf{x}') = \frac{y_l(\mathbf{x}') + y_r(\mathbf{x}')}{2} \quad (4)$$

Finally, the decision maker will determine the class label by comparing the crisp output against the threshold.

Since in this work type-2 FRBC will be compared against type-1 FRBC, it is appropriate to briefly highlight the differences between both classifiers. The structure of a type-1 FRBC is similar to a type-2 FRBC except for a few aspects. Firstly, the inference engine will produce a firing strength, f^j for j th rule rather than an interval value. Secondly, the type-reducer does not exist since no type-2 number is involved. In other words, the output processing only consists of defuzzification. For height defuzzification, the crisp output, y can be computed as:

$$y(\mathbf{x}') = \frac{\sum_{j=1}^T y^j f^j}{\sum_{j=1}^T f^j} \quad (5)$$

3 Training of Fuzzy Classifier

The tuning of fuzzy classifiers with GAs have been pioneered by Valenzuela-Rendón [7] where GAs are used to select fuzzy rules. Since then, GAs have been successfully applied in the tuning of FRBCs. One of the most interesting problems in GAs is coding the solution space. In this work, the Pittsburgh approach is used where each chromosome encodes the whole rule base, and the best chromosome at the end of the evolution determines the winning FRBC. The

classification accuracy of the classifier with the respective rule base is used as the fitness function. The chromosome encoding is straightforward. A complete chromosome that represents a type-2 FRBC rule base has five parts as shown in Fig. 2. The first part encodes the upper membership function (UMF) parameters. A Gaussian MF needs two parameters—mean and standard deviation while a triangular MF needs three parameters—left, apex and right points. Assuming that global fuzzy rules (non rule-specific MFs) are adopted and each feature is partitioned into q fuzzy sets, then the number of genes in this part is equal to $p \times q \times z$ where z is the number of parameters required for a MF (i.e., 2 for Gaussian MF and 3 for triangular MF). Next, the second part represents the consequent labels, $L \in \{1, \dots, K\}$ where K denotes the total number of classes. The length of this part is equal to the number of rules, $M = q^p$. The third part defines the ratio of lower membership function (LMF) height to UMF height which is in the range of $(0, 1]$. This part has $p \times q$ genes. The next part comprises of genes with length of M which characterises the rule flags to control whether a rule should be ignored (flag = ‘0’) or included in the rule base (flag = ‘1’). This is known as rule pruning method. Later in section 4.2, chromosome which represents full rule base fuzzy classifier will have non-evolvable flags in which every flag is preset to ‘1’. Finally, the last part represents a set of thresholds to divide the crisp outputs into discrete classes. It has length of $(K - 1)$. For example, a two-class problem requires a threshold at interval $[1, 2]$ while a three-class problem requires two thresholds: one at interval $[1, 2]$ and another at interval $(2, 3]$. As such, a complete type-2 FRBC chromosome has $(2M + pq(1 + z) + k - 1)$ genes. All genes are binary coded in the current framework. For parts “MF parameters” and “threshold” where continuous values are required in the phenotype space, each gene from these parts is encoded as a 8-bit string. During the fitness evaluation, the parameters are decoded into real numbers using linear mapping equation as shown below:

$$g = G_i^{min} + (G_i^{max} - G_i^{min}) \times \frac{A_i}{2^8 - 1} \quad (6)$$

where g denotes the actual value of the i^{th} parameter, A_i denotes the integer represented by a 8-bit string gene, G_i^{max} and G_i^{min} denote the user defined upper and lower limits of the gene respectively. For the remaining parts of the chromosomes, the binary encoded genes are directly decoded into integer numbers. The selection method is tournament with size of two. The elitist strategy is used to ensure that the best chromosome (the one with the highest fitness) always survives in the next generation. According to [8], simple GA models should be tried first and be dismissed only if they do not provide a satisfactory result. Thus, the genetic operators in this work— bitwise flipping mutation and single-point crossover, are kept as simple as possible and yet achieving good solutions. The mutation rate is kept

MF Parameters	Class Labels	Ratios	Flags	Thresholds

Fig. 2. The structure of a chromosome

relatively low (0.03) while the crossover rate is set moderately high (0.8) to keep a good balance between exploration and exploitation and thus avoiding premature convergence. In section 4, the population size is set at 50 and the maximum number of generations is fixed at 200. The optimization process stops if there is no improvement in the fitness functions of the past 30 generations.

4 Application to Ford Automotive Dataset

The proposed fuzzy rule-based classifiers have been applied to Ford automotive dataset [9]. In this real world application, the classifier needs to detect the presence of a human in a vehicle. One possible scenario would be that when a driver returns to his or her vehicle at night, particularly in a deserted location, the knowledge that no one is hiding inside the vehicle can provide peace of mind.

Raw analog signals were collected from a vibration sensor which is located at the vehicle's suspension system. The signals are then filtered by a low pass filter (LPF) and converted into digital signals [10]. Each diagnostic session has 500 sample points. The length of the sequences reflects the time available for making the classification decision. Presumably, the task would be easier if the sequence length were increased, but this would violate the requirements of the application. The beginning of the sampling process is not aligned with any external circumstance or any aspect of the observed pattern. The training data (3306 samples) were collected under typical operating condition with minimum noise but the testing data (810 samples) were collected under noisy conditions such as wind disturbances.

4.1 Feature Extraction Method

The problem does not appear to have a simple solution that emerges from visual inspection of these data sequences as shown in Fig. 3(a). A signal processing tool such as periodogram may be useful to reveal any interesting features. In

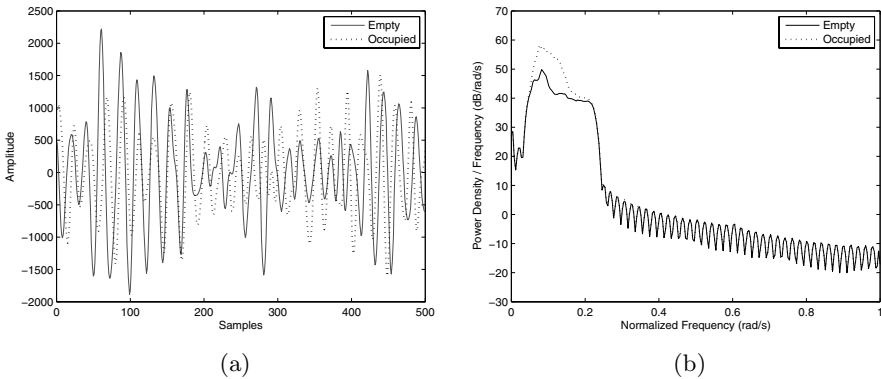


Fig. 3. (a) Vibration signals , (b) average periodogram of the training samples

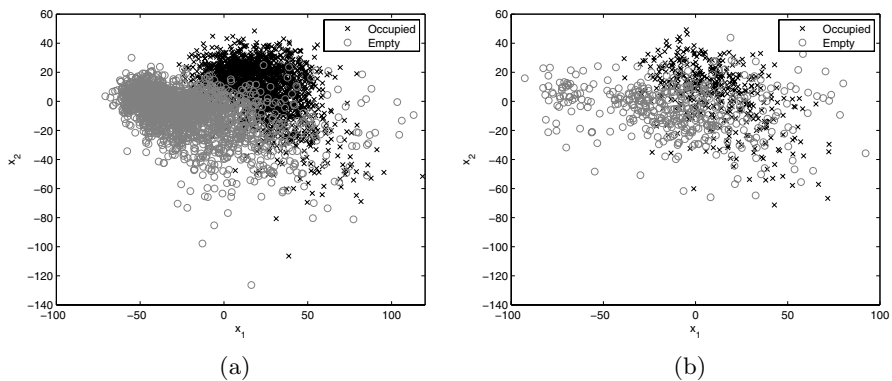


Fig. 4. 2-D scatter plots of PCA projected (a) train data, (b) test data

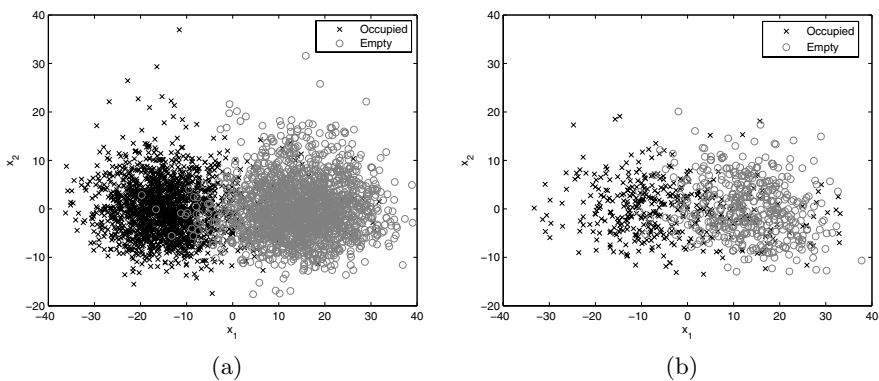


Fig. 5. 2-D scatter plots of LDA projected (a) train data, (b) test data

this application, the periodogram are computed with 512-point FFT and triangle window. Thus, the periodogram is a coefficient vectors with length of 257. The average periodograms of the training samples are shown in Fig. 3(b). The figures show that the discriminative features are mostly located at the low frequency regions. If each of the periodogram coefficient is regarded as an input dimension, then the total number of feature dimensions is 257 which is impractical for most classifiers. Therefore, it is necessary to reduce the feature dimensions to lower dimensions. PCA is one of the most popular feature dimensionality reduction techniques. This technique searches for directions in the data that have largest variance and subsequently project the data onto it. However, it is completely unsupervised, knows only about variance, but nothing about different classes of data. In light of this, LDA may reveal class structure better. This technique maximizes the ratio of between-class variance to the within-class variance in any particular data set thereby guaranteeing maximal separability. Fig. 4 and 5 show the two-dimensional scatter plots where the features are extracted with PCA and LDA respectively. It is clear that the variance within classes is smaller

and the variance between classes is bigger in LDA projection. The feature space produced by LDA is more linearly separable while PCA gives less optimal separation between two classes especially on the noisy test data. As a result, the data produced by PCA require a more sophisticated classifier in order to handle the blurred decision boundary. In contrast, LDA projected data impose a less stringent requirement on the classifiers. Thus, it would be interesting to investigate if a more advanced classifier like type-2 FRBC can perform better than type-1 FRBC when ambiguity in the feature extraction process becomes a source of uncertainty.

4.2 Performance Evaluation

Four FRBCs are proposed to examine if type-2 FRBC can outperform type-1 counterpart when the decision boundary is imprecise. They are type-1 FRBCs with full and pruned rule base (T1-FRBC(F), T1-FRBC(P)) and type-2 FRBCs with full and pruned rule base (T2-FRBC(F), T2-FRBC(P)) also. Each kind of classifier consists of 10 different designs in which they are evolved with GAs separately. The performance metrics used in the evaluation are the accuracy of the classifier and the false positive rate (FPR). False positive occurs when the classifier reports that the vehicle is occupied when no one is actually in it. In particular, the systems can be sensitive to false positives in windy conditions. Table 1 shows that FRBCs with pruned rule base generally perform much better. This is consistent with the findings in [11] whereby it was demonstrated that the decision boundary of a winner-takes-all based T1-FRBC with complete rule base is rectangle or hyperrectangle. Although the proposed T1-FRBC(F) and T2-FRBC(F) in this paper will not have completely rectangular boundaries, the boundaries near the edge of the feature space is still parallel to the feature axes. Conversely, at the middle region the decision boundary can be non-linear due to the rules aggregation effect of defuzzification and type-reduction. As such, the proposed FRBCs with complete rule base may not classify the PCA projected data (see Fig. 4(a)) very well near the edges because these regions require boundary that is not in-parallel with the feature axes. This shows the limitation of the full rules FRBCs themselves. On the other hand, FRBC with pruned rule base does not suffer from this issue. In addition, the results show that T1-FRBC(F) has the worst performance whereas T2-FRBC(P) is clearly the winner. It is interesting to see how well the T2-FRBC(F) fares against the FRBCs with pruned rule base although it has the aforementioned limitation. In fact, it achieves the highest test accuracy and the lowest FPR. From Table 2, the results show an overall improvement on both classification accuracy and false positive rate. This is not surprising as the LDA does a better job than PCA in partitioning the data from both classes. Fig. 5(a) indicates that the data can be classified well with boundary which is in-parallel with feature axes. Thus, all FRBCs regardless of full or pruned rule base are free from limitations and both type-1 and type-2 FRBCs can perform just equally well.

Table 1. Average and Standard Deviation of Classification Accuracy and False Positive Rate Across 10 Iterations with PCA Based Feature Extraction

Classifier	Dataset	Average ACC (%)		Average FPR (%)	
		Mean	Stdv	Mean	Stdv
T1-FRBC(F)	Train	90.90	1.83	6.87	2.94
	Test	72.27	3.06	25.02	5.10
T1-FRBC(P)	Train	92.63	0.70	4.85	1.23
	Test	73.99	1.43	23.66	2.47
T2-FRBC(F)	Train	92.08	0.64	5.44	1.09
	Test	74.19	1.16	22.75	1.69
T2-FRBC(P)	Train	92.96	0.47	4.97	1.22
	Test	74.14	1.28	23.98	2.68

Table 2. Average and Standard Deviation of Classification Accuracy and False Positive Rate Across 10 Iterations with LDA Based Feature Extraction

Classifier	Dataset	Average ACC (%)		Average FPR (%)	
		Mean	Stdv	Mean	Stdv
T1-FRBC(F)	Train	95.49	0.00	3.10	0.00
	Test	81.04	0.06	7.14	0.17
T1-FRBC(P)	Train	95.47	0.05	3.27	0.41
	Test	81.05	0.47	7.56	0.88
T2-FRBC(F)	Train	95.50	0.02	3.09	0.02
	Test	81.01	0.05	7.21	0.13
T2-FRBC(P)	Train	95.48	0.04	3.35	0.57
	Test	81.25	0.49	7.50	0.85

5 Conclusion

In this paper, genetically evolved FRBCs are used to analyse if type-2 framework can help in the case of poor feature selection through its FOU. The observations from the automotive application above show that type-2 FRBC may excel when the decision boundary is imprecise and the fuzzy classifier itself does not have enough degrees of freedom to construct a suitable boundary. When the classification task becomes easier or the FRBCs are given enough degrees of freedom then the advantage of type-2 framework may no longer be significant anymore. This probably explains why in [3] both type-1 and type-2 FRBCs have close performances. Nevertheless, using a type-2 FRBC in real world classification problems can be a better choice than type-1 FRBC since the amount of uncertainty in real problem most of time is difficult to estimate. As a rule of thumb, type-2 FRBC performance is at least comparable, if not better than type-1 FRBC. Hence, for

applications where the computation speed is not a major consideration, type-2 framework should be adopted.

References

1. Mendel, J.: Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions. Prentice Hall, Upper Saddle River (2001)
2. Starczewski, J.: What Differs Interval Type-2 FLS from Type-1 FLS? In: Rutkowski, L., Siekmann, J.H., Tadeusiewicz, R., Zadeh, L.A. (eds.) ICAISC 2004. LNCS, vol. 3070, pp. 381–387. Springer, Heidelberg (2004)
3. Wu, H., Mendel, J.: Classification of Battlefield Ground Vehicles Using Acoustic Features and Fuzzy Logic Rule-Based Classifiers. *IEEE Trans. on Fuzzy Systems* 15(1), 56–72 (2007)
4. Ravi, V., Reddy, P.J., Zimmermann, H.-J.: Pattern Classification with Principal Component Analysis and Fuzzy Rule Bases. *European Journal of Operational Research* 126, 526–533 (2000)
5. Chua, T.W., Tan, W.W.: GA Optimisation of Non-Singleton Fuzzy Logic System for ECG Classification. In: *Proceeding of IEEE Congress of Evolutionary Computation*, pp. 1677–1684 (2007)
6. Mendel, J., Hagaras, H., John, R.I.: Standard Background Material About Interval Type-2 Fuzzy Logic Systems That Can Be Used By All Authors. *IEEE Computational Intelligence Society*
7. Valenzuela-Rendón, M.: The Fuzzy Classifier System. Motivations and First Results. In: Schwefel, H.-P., Männer, R. (eds.) *PPSN 1990*. LNCS, vol. 496, pp. 338–342. Springer, Heidelberg (1991)
8. Kuncheva, L.I.: *Fuzzy Classifier Design*. Physica-Verlag, New York (2000)
9. Ford Classification Challenge, http://home.comcast.net/~nn_classification/
10. Eagen, et al.: System and Method for Detecting Presence of a Human in a Vehicle. U.S. Patent No. 7,353,088 B2 (2008)
11. Ishibuchi, H., Nakashima, T.: Effect of Rule Weights in Fuzzy Rule-Based Classification Systems. *IEEE Trans on Fuzzy Systems* 9(4), 506–515 (2001)

Improving XCS Performance by Distribution

Urban Richter, Holger Prothmann, and Hartmut Schmeck

Karlsruhe Institute of Technology – Institute AIFB

76128 Karlsruhe, Germany

{urban.richter,holger.prothmann,hartmut.schmeck}@kit.edu

Abstract. Learning Classifier Systems (LCSs) are rule-based evolutionary reinforcement learning (RL) systems. Today, especially variants of Wilson’s eXtended Classifier System (XCS) are widely applied for machine learning. Despite their widespread application, LCSs have drawbacks: The number of reinforcement cycles an LCS requires for learning largely depends on the complexity of the learning task. A straightforward way to reduce this complexity is to split the task into smaller sub-problems. Whenever this can be done, the performance should be improved significantly. In this paper, a nature-inspired multi-agent scenario is used to evaluate and compare different distributed LCS variants. Results show that improvements in learning speed can be achieved by cleverly dividing a problem into smaller learning sub-problems.

1 Introduction

The complexity of today’s technical systems is continuously increasing. Future systems will consist of a multitude of autonomous soft- and hardware components that interact with each other to satisfy functional requirements of the global system. Since this trend bears the risk of unpredictable or even uncontrollable system behaviour, Organic Computing (OC)¹ focuses on monitoring, analysing, and controlling complex distributed systems to endow them with the ability of *controlled self-organisation*, which means that an organic system can self-organise to achieve its tasks while being simultaneously observed and – if necessary – influenced by a higher level component to avoid unwanted emergent system states. To this end, an observer/controller architecture has been proposed in [1] that monitors and analyses the status of a System under Observation and Control (SuOC) and influences its behaviour (cf. Fig. 1). Since it is in general impossible

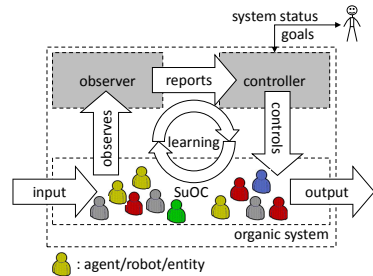


Fig. 1. Observer/controller architecture

¹ We gratefully acknowledge the financial support by the German Research Foundation (DFG) within the priority programme 1183 Organic Computing.

to foresee all possible configurations the SuOC takes on in a dynamic environment, it is important to endow the observer/controller with learning components so that it can learn autonomously new control actions for unforeseen situations and evaluate their consequences.

Learning Classifier Systems (LCSs) are rule-based evolutionary on-line learning systems that are suitable to perform the learning task in an observer/controller component. Here, they are used to learn control rules for a nature-inspired multi-agent scenario. A special focus is on the use of distributed LCSs that share the decision task and their comparison to a single LCS. The remainder of this paper is structured as follows: Section 2 gives an introduction to LCSs and previous work on distributed LCS architectures is summarised in Sect. 3. The nature-inspired multi-agent scenario used as test scenario in this work is presented in Sect. 4. Section 5 describes the application of a single LCS for the learning task while distributed variants are discussed and compared in Sect. 6. The paper concludes with a summary and a discussion of future work in Sect. 7.

2 Learning Classifier Systems (LCSs)

LCSs are rule-based on-line learning systems that combine nature-inspired optimisation heuristics and reinforcement learning (RL) techniques to learn appropriate actions for any input they get [2]. They are applicable to any problem where a numerical reward reflecting the quality of an action can be obtained. The core component of an LCS is its rule base that contains rules (called *classifiers*) consisting of a condition, an action, and a prediction. The selection of an appropriate action for a given input is a two-step process. From the rule base of all classifiers a subset called *match set* is computed that contains all classifiers whose condition matches the current input. For all distinct actions present in the match set the average prediction of all classifiers advocating that action is calculated. The action with the highest average prediction is selected for execution and all classifiers in the match set advocating that action form the *action set*. The reward received from the environment is subsequently used to update the predictions of all classifiers in the action set.

Classifiers forming the rule-base are created in two different ways: Whenever the match set is empty, classifiers consisting of a condition matching the current input, a random action, and a default prediction are inserted into the rule base by a process called *covering*. Furthermore, occasionally, a randomised *evolutionary component* selects relatively good classifiers to be the parent individuals for a reproduction cycle. Crossover and mutation are applied to copies of the parents to form offspring which are inserted into the rule base.

A variety of different LCS implementations has been proposed (cf. [3]), many are based on Wilson's XCS [2] system, which is an LCS implementation that maintains separate prediction and fitness values and where the fitness of a classifier is based on the accuracy of the prediction reward. While Wilson originally used a binary input encoding for XCS, different approaches to represent real-valued inputs have been examined (e.g., [4]).

3 Related Work on Distributed LCSs

Parallel and hierarchically arranged LCSs have been investigated to reduce the number of reinforcement cycles necessary for learning a given task. However, the problem of increased convergence times in the case of big search problems still seems to be not fully solved. An overview of work that has been done before 1996 is available in [5], more recent work is briefly reviewed here.

An idea to enhance the learning speed of complex simulated robot behaviours is presented in [6]. Five hierarchically arranged LCSs are used, where three LCSs learn basic robot behaviours (chasing, avoidance, and escaping) and two other LCSs combine these basic behaviours to more complex behaviours. The work seems similar to work presented before by Dorigo [7]. Dorigo investigated LCSs as a tool for building adaptive control systems for real robots showing that it is possible to let the *AutonoMouse*, a small real robot, learn to approach a light source under a number of different noise and lesion conditions.

In [8] an XCS is used in combination with a client/server architecture for distributed data mining. Each client has its own XCS that learns classifications based on a local database and reports the learned classification model to the server. The server aggregates the models to its own XCS and additionally learns all unsolved problems that could not be solved locally by the clients. This distributed XCS architecture is compared to a single XCS implementation and the performance evaluation shows that the distributed XCS is competitive to the single XCS when the noise level is small. When the level of noise in the data is increased, the distributed XCS needs more time to recover.

In [9] a similar idea is proposed. The authors try to improve global XCS performance by solving smaller sub-problems and combining the solutions. Results are demonstrated on the binary coded multiplexer problem, a common test environment for the XCS.

To solve a global problem and improve the learning speed, all cited papers divide a problem into sub-problems, learn each of the lower level problems, and combine the sub-results into a global result. Whenever this can be done, the performance is improved. However, often the identification of sub-problems that can be treated separately is a non-trivial task and different sub-problem formulations influence the performance of the distributed system significantly. Here, we investigate different ways of modelling a distributed LCS for a nature-inspired multi-agent scenario and evaluate their performance.

4 Experimental Environment

The multi-agent system described below (cf. Fig. 2) and used for the experimental validation of the different LCS implementations is inspired by nature. The simulation reproduces the collective cannibalistic behaviour of densely packed chickens in cages and tries to detect, predict, and/or prevent the unwanted behaviour of clustering, which occurs frequently when a chicken is wounded, and which leads to a major loss of animals (up to 50% of the animals). If a wounded

chicken is perceived, this chicken is chased and picked on by the others until it dies [10]. Chasing and picking wounded chickens leads to the emergent formation of chicken swarms (or clusters). A swarm disperses when the wounded chicken is killed. The emergent behaviour is spatial, but swarms move over time. This is a case of *negative*, i. e., undesired, emergence, due to interactions between the individual chickens.

Currently, the chicken cages are supervised by humans, their task is to prevent chicken deaths by using appropriate disturbing actions. Viewed as an OC scenario, the objective is to observe, classify, and control (global and macroscopic) emergent behaviour automatically. To achieve this goal and reduce the chicken death rate, we use the observer/controller paradigm as introduced in detail in [1,10] to report a quantified context of the underlying system to the controller, which evaluates the situation and reacts with adequate control actions to disperse chicken swarms or to prevent their formation. Instead of a real chicken a simple model of chicken behaviour is used, in simulation order patterns emerge as expected in form of chicken swarms.

In the simulation, every chicken can move to eight different directions (north, north-east, east, south-east, south, south-west, west, and north-west) at the speed of one movement per simulation tick. A noise signal with variable intensity i and duration d can be applied at an arbitrary position in the cage to frighten the chickens and scare them off, which leads to the dispersion of a possibly existent cluster. Moreover, the noise signal also has a negative effect: If the noise is too loud or occurs for too long then chickens are killed (eventually) by noise intervention. Therefore, it is not beneficial to continuously apply a noise signal to prevent clusters.

In the simulated environment, a chicken is characterised by the attributes heading, position (x -, y -coordinates), and energy/vital force. It is directed by a predefined fixed finite state machine and will be influenced by the behaviour of other chickens in its local neighbourhood or by changes in the environment, e. g., noise that frightens the chickens. Chickens are considered as autonomous robots or agents with simple rules and local goals, they aim for surviving as long as possible, and they are attracted by wounded agents.

We observe a scenario of 40 chickens that move randomly in a playground (the cage) of 30×30 fields. When a chicken is killed, a new chicken is generated and placed randomly in the cage. The observer collects the raw data of the chickens and computes additionally to the so-called *emergence values* e_x of the x -coordinates, e_y of the y -coordinates, and e_h of the heading², the centroid (x_c, y_c) of the chicken population.

The emergence values characterise the whole system as a result of a transformation process of the raw data to some higher abstract metrics and correspond to the deviation of the maximum of Shannon's information theoretical entropy values for each observed attribute, as described in more detail in [11].

² The emergence of the energy value is not taken into account, since it is an intrinsic property of the agents, which is not available to an external observer.

Consequently, the situation parameters given to the controller are combined to a vector $S_t = (e_x, e_y, e_h, (x_c, y_c))$. The actions taken by the controller have the form $A = (d, i, (x_c, y_c))$, the centroid (x_c, y_c) of the chicken population has already been determined by the observer.

5 Learning with a Single LCS

Without noise interventions approximately 336 chickens are killed in a simulation with 10 000 ticks. As described in [10], applying a static noise signal with fixed intensity i and fixed duration d around the computed population centroid (x_c, y_c) to frighten the chickens and disperse the cluster works well and decreases the death rate. In doing so, the success of noise interventions depends on the chosen parameters of duration and intensity, as investigated in [12]. Further death rate reductions can be achieved by integrating a learning process that adapts quickly the intensity i and the duration d of the noise signals to the measured emergence values.

In [12] we have modified the XCSJava1.0 reference implementation by Butz [13] to work with real-valued inputs and simulated the chicken scenario with 20 different random seed values and a maximal population of 800 classifiers to compute the average number of killed chickens $\#kC$. As reward – to compute the fitness of a classifier – we monitor (and the XCS minimises) the number of killed chickens $\#kC$ during and after an action $A_t = (d_t, i_t)$ before the next action $A_{t+1} = (d_{t+1}, i_{t+1})$ occurs (where t is the simulation time). The LCS learns the best combination of the three mentioned emergence values and the two parameters d and i of the noise signal.

Using this central/single learning loop, the following results were obtained [12]: The XCS adapted slowly and many evaluation loops had to be made before the XCS converged with a good condition-action-mapping. We varied the size of the action space from 15 ($d \in \{5, 6, \dots, 9\} \times i \in \{0, 10, 20\}$) up to 135 ($d \in \{1, 2, \dots, 15\} \times i \in \{0, 10, 20, \dots, 80\}$) different control actions, where every combination of d and i is possible.

The results confirmed that the XCS needs more and more time when the number of options of the condition-action-mappings increases. Therefore, the research question of the following section is: Can we improve the global XCS performance by splitting the objective space (characterised by the options of condition-action-mappings) into smaller sub-problems and by solving/combining these smaller sub-problems with parallel or hierarchically ordered LCSs that solve the global problem collaboratively?

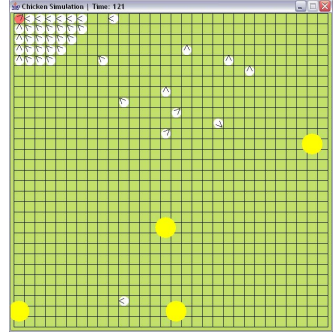


Fig. 2. Snapshot of the chicken simulation: Healthy chickens are white, wounded chickens are dark (red), and feeding troughs are shown as bigger circles

6 Distributed Learning

To make use of LCSs as a fast on-line learning mechanism that can be integrated into the general observer/controller architecture, improvements are necessary. An LCS architecture is needed that (i) needs less RL loops to learn while (ii) exhibiting the same or better learning results compared to existing LCSs. Solutions to this challenge might include: (a) Parallel or distributed learning architectures: By dividing the problem into sub-problems and allowing various LCSs to work together and run in parallel, the complete learning task is decomposed and improvements in learning speed should be possible. (b) Algorithmic advancements: Covering techniques and genetic operators like crossover and mutation could be varied and improved. (c) More training phases before acting on real-time problems: In the case of OC systems that should adapt continuously in dynamic environments more training phases do not seem to be an adequate strategy because changes occur on-line and adaptation has to cope with these changes in real-time. In this paper, we concentrate on improvements through parallelism. Algorithmic changes are future work.

6.1 Parallel Learning (2PXCS)

A promising way to deal with complex problems should be to divide them into sub-problems and assign each sub-problem to a single LCS instance. In this way, each single LCS learns to solve a specific sub-task and the sub-task results can be combined resulting in a parallel learning system that performs as well as or hopefully better than a single LCS.

The design approach of task decomposition (as presented in the following) is not provided in an automated way. Depending on the problem, a decomposition might be difficult or even impossible due to strong interdependencies of the sub-tasks. The system designer needs to identify independent and basic subtasks that could be assigned to different LCS instances.

As depicted in Fig. 3(b), we replace the single XCS (cf. Fig. 3(a)) by two completely separated parallel learning loops – one responsible for mapping the three emergence values to the duration of the noise intervention, the other one responsible for finding the best intensity of the intervention. The two parallel LCSs use the same standard parameters as given by Butz [13], get the same reward as mentioned in Sect. 5, and both have a four dimensional search space. We

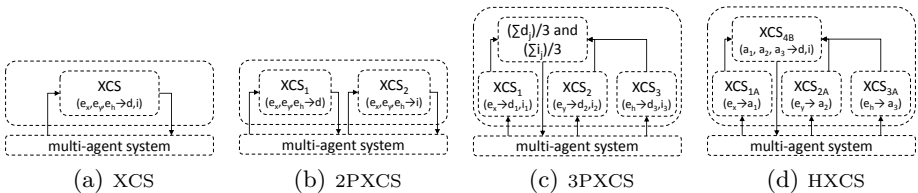


Fig. 3. Single vs. distributed architectures

call this scenario 2PXCS for two parallel XCS instances. Referring to Fig. 3(b), XCS_1 maps the emergence values e_x , e_y , and e_h to the duration d , XCS_2 does the same with the intensity i . We run again our experiments with two different configurations with 15 ($d \in \{5, 6, \dots, 9\} \times i \in \{0, 10, 20\}$, cf. Fig. 4(a)) and 90 ($d \in \{1, 2, \dots, 15\} \times i \in \{0, 10, \dots, 50\}$, cf. Fig. 4(b)) possible combinations of the two parameters d and i and compare them to the results of the corresponding single XCS.

Figures 4(a) and 4(b) depict the average number of killed chickens $\#kC$ per 1 000 simulation ticks over 20 runs. We can see that both, the single XCS and the 2PXCS implementation, improve their condition-action-mappings over time and converge finally to a steady result. Comparing the scenario with 15 different actions (Fig. 4(a)) and the scenario of 90 possible actions (Fig. 4(b)), we note that in the first case the 2PXCS shows a better performance than the single XCS while in the second scenario the single XCS outperforms the parallel variant.

In the 2PXCS variant, XCS_1 and XCS_2 have to learn a four dimensional condition-action-mapping, i. e., that is their condition-action-mapping is reduced by one dimension compared to single XCS. The results are contradictorily and a closer view on the learned classifiers in the logged population files shows that all classifiers have a high prediction error, indicating that a bad condition-action-mapping has been learned. Therefore, we conclude that the parameters duration d and intensity i are not sufficiently independent to be learned by two separated LCS instances. Thus, we tried another – more reasonable – parallel learning architecture as described in the following.

6.2 Learning with Three Parallel LCSs (3PXCS)

As depicted in Fig. 3(c), we divide the single XCS (cf. Fig. 3(a)) into three separated parallel learning loops – every LCS is responsible for mapping one of the three emergence values to the duration and the intensity of the noise intervention, i. e., every XCS has to learn a three dimensional condition-action-mapping. We call this scenario 3PXCS because of the three parallel XCS instances.

The LCSs again use the same standard parameters as proposed by Butz [13]. Every XCS_j with $j \in \{1, 2, 3\}$ determines in every RL loop one specific noise signal (d_j, i_j) based on which we compute the average proposed duration $\bar{d}$ with $\bar{d} = \frac{1}{3} \cdot \sum_{j=1}^3 d_j$ and the average intensity $\bar{i}$ with $\bar{i} = \frac{1}{3} \cdot \sum_{j=1}^3 i_j$. The triggered noise signal is the average $(\bar{d}, \bar{i})$ of all three decisions and the reward every XCS_j receives is the reward received under the control action $(\bar{d}, \bar{i})$.

Figures 5(a) and 5(b) depict the average number of killed chickens $\#kC$ per 1 000 simulation ticks over 20 runs. We can see that the 3PXCS implementation improves the condition-action-mappings over time and finally, converges to a steady result.

Comparing the scenario with 15 actions (Fig. 5(a)) and the scenario of 90 actions (Fig. 5(b)), we see consistent results. The new design decision to learn both parameters of a noise signal in one single LCS and to simplify the condition part of a classifier seems to be the better decision in this case. However, the results outperform the results we made with the single XCS.

6.3 Hierarchical Learning (HXCS)

Based on the results we made with the parallel implementations and the fact that 3PXCS nearly loses its advantage compared to the single XCS after 800 000 simulation ticks in 90 actions scenario, we developed a hierarchically organised LCS architecture, as depicted in Fig. 3(d). We call this type of architecture HXCS.

On Level A we implemented three LCSs that work parallel on different sub-problems. XCS_{1A} maps only the emergence value e_x to the decision/action $a_1 \in \{0, 1\}$ if a noise intervention is preferred or not. XCS_{2A} and XCS_{3A} do the same with e_y and e_h , respectively. The three LCSs on Level A have to learn a binary coded decision, where 0 means no noise signal should be applied and 1 represents a noise intervention.

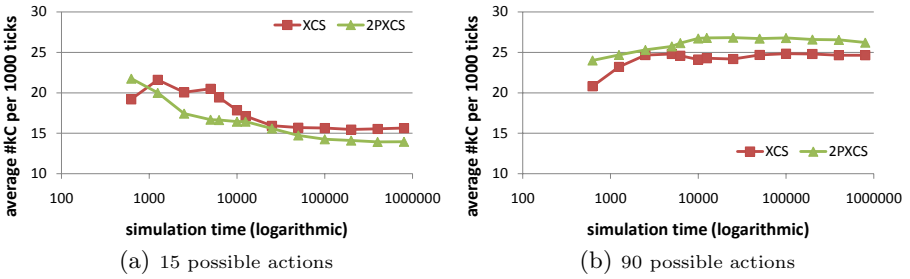


Fig. 4. Learning over time: XCS vs. 2PXCS, averaged values over 20 runs

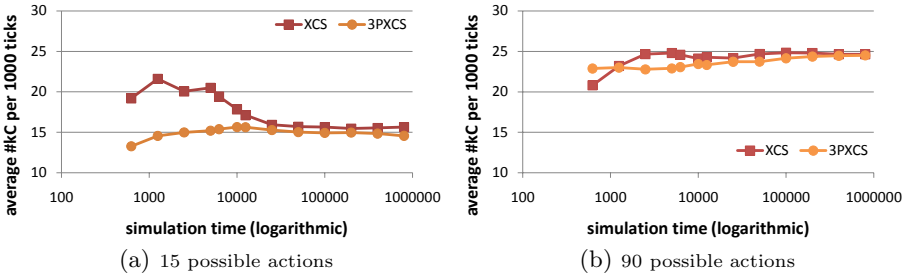


Fig. 5. Learning over time: XCS vs. 3PXCS, averaged values over 20 runs

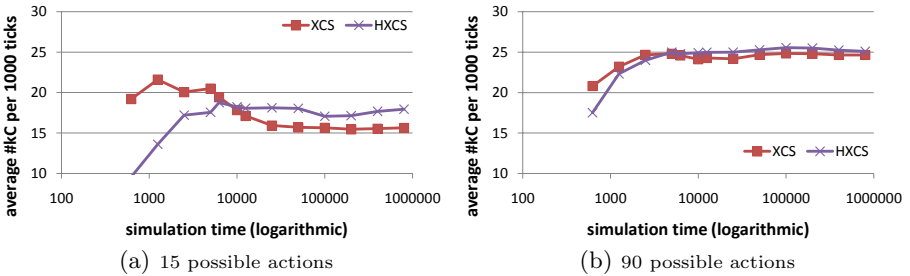


Fig. 6. Learning over time: XCS vs. HXCS, averaged values over 20 runs

A fourth LCS, XCS_{4B} , collects the decisions of the Level A LCSs, XCS_{1A} , XCS_{2A} , and XCS_{3A} , and maps this input data to a noise signal that is again characterised by a duration d and an intensity i – while a so called null-action (d, i) with $d \in \{5, \dots, 9\}$ and $i = 0$ is explicitly possible on Level B. We should mention that a null-action is equal to the effect of intervening the chickens without any noise.

XCS_{1A} , XCS_{2A} , and XCS_{3A} have to solve a sub-problem with a two dimensional condition-action-mapping, XCS_{4B} has to solve a five dimensional problem where the situation is characterised as a 3 bit binary vector.

As done before, we compare the hierarchically organised HXCS implementation with the single XCS in the case of two configuration scenarios. The results are presented in Fig. 6(a) and 6(b). We see that HXCS improves its condition-action-mappings over time and finally, converges to a steady result. But, we cannot demonstrate significant advantages of this new LCS arrangement. To the contrary, HXCS shows no improvements in learning speed and quality of the results. We conclude that the condensed situation information available to XCS_{4B} is too heavily simplified. Instead of easing the learning task, it seems difficult to decide on noise interventions based on the pre-processed data provided by Level A.

7 Conclusion and Outlook

Aiming at enhancing the learning speed of LCSs, we focused especially on the XCS implementation provided by Butz [13], which has been modified to work with real-valued parameters. Caused by drawbacks LCSs have in learning speed our experimental results validate – at least partially – the idea of distributed organised LCSs that solve smaller sub-problems of a larger problem. We used LCSs to learn the best control intervention on-line in a nature-inspired multi-agent scenario and compared differently arranged XCS implementations. Improvements in the performance and learning speed could be shown specially for the 3PXCS implementation. The other investigated architectures (2PXCS and HXCS) started with a promising idea, but exhibited no convincing results in the investigated scenario.

We conclude that improvements in learning speed could be achieved by dividing a problem into sub-problems that are solved in parallel. However, identifying appropriate sub-problems is a difficult task that – depending on the problem – might even be impossible. Future work will focus on algorithmic advancements in LCSs. Creating new classifiers, deleting classifiers with a high prediction error, or running genetic operators like crossover and mutation are methods that can be varied and optimised in the case of a special learning problem.

References

1. Richter, U., Mnif, M., Branke, J., Müller-Schloer, C., Schmeck, H.: Towards a generic observer/controller architecture for Organic Computing. In: INFORMATIK 2006 – Informatik für Menschen!, pp. 112–119. Köllen Verlag (2006)

2. Wilson, S.W.: Classifier fitness based on accuracy. *Evolutionary Computation* 3(2), 149–175 (1995)
3. Kovacs, T.: Learning classifier systems resources. *Soft Computing* 6(3–4), 240–243 (2002)
4. Dam, H.H., Abbass, H.A., Lokan, C.: Be real! XCS with continuous-valued inputs. In: *Proceedings of the 2005 Workshops on Genetic and Evolutionary Computation (GECCO 2005)*, pp. 85–87. ACM, New York (2005)
5. Barry, A.: Hierarchy formation within classifier systems – A review. In: *Proceedings of the 1st International Conference on Evolutionary Algorithms and their Applications (EVCA 1996)*, Moscow, pp. 195–211 (1996)
6. Baneamoon, S.M., Salam, R.A., Talib, A.Z.H.: Learning process enhancement for robot behaviors. *Int. Journal of Intelligent Technology* 2(3), 172–177 (2007)
7. Dorigo, M.: Alecsys and the autnomouse: Learning to control a real robot by distributed classifier systems. *Machine Learning* 19(3), 209–240 (1995)
8. Dam, H.H., Abbass, H.A., Lokan, C.: DXCS: An XCS system for distributed data mining. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2005)*, pp. 1883–1890. ACM, New York (2005)
9. Gershoff, M., Schulenburg, S.: Collective behavior based hierarchical XCS. In: *Proceedings of the 2007 Genetic And Evolutionary Computation Conference (GECCO 2007)*, pp. 2695–2700. ACM, New York (2007)
10. Mnif, M., Richter, U., Branke, J., Schmeck, H., Müller-Schloer, C.: Measurement and control of self-organised behaviour in robot swarms. In: Lukowicz, P., Thiele, L., Tröster, G. (eds.) *ARCS 2007. LNCS*, vol. 4415, pp. 209–223. Springer, Heidelberg (2007)
11. Mnif, M., Müller-Schloer, C.: Quantitative emergence. In: *Proceedings of the 2006 IEEE Mountain Workshop on Adaptive and Learning Systems (IEEE SMCals 2006)*, pp. 78–84 (July 2006)
12. Richter, U., Mnif, M.: Learning to control the emergent behaviour of a multi-agent system. In: *Proceedings of the 2008 Workshop on Adaptive Learning Agents and Multi-Agent Systems at AAMAS 2008 (ALAMAS+ALAg 2008)*, pp. 33–40 (May 2008)
13. Butz, M.V.: XCSJava 1.0: An implementation of the XCS classifier system in Java. Technical Report 2000027, Illinois Genetic Algorithms Laboratory (2000)

Evolving an Ensemble of Neural Networks Using Artificial Immune Systems

Bruno H.G. Barbosa^{1,2}, Lam T. Bui², Hussein A. Abbass²,
Luis A. Aguirre¹, and Antônio P. Braga¹

¹ Department of Electronic Engineering, Federal University of Minas Gerais,
Belo Horizonte, MG, Brazil

{brunohb, aguirre, apbraga}@cpdee.ufmg.br

² School of Information Technology and Electrical Engineering, Australian Defence
Force Academy, University of New South Wales, Canberra, ACT, Australia
lam.bui07@gmail.com, h.abbass@adfa.edu.au

Abstract. This paper presents a novel ensemble construction approach based on Artificial Immune Systems (AIS) to solve regression problems. Over the last few years AIS have increasingly attracted interest from researchers due to their ability to balance the exploration and exploitation of the search space. Nevertheless, there have been just a few applications of those algorithms in the construction of committee machines. In this paper, a population of feed-forward neural networks is evolved using the Clonal Selection Algorithm and then ensembles are automatically composed of a subset of this neural network population. Results show that the proposed algorithm can achieve good generalization performance on some hard benchmark regression problems.

Keywords: Neural network, ensemble, regression, immune artificial systems, clonal.

1 Introduction

In the last decades, Supervised Machine Learning algorithms, a research area that focuses on the study of computer algorithms to automatically extract information from data sets [1], have emerged as an important subject in computer science. The purpose of these algorithms is to improve the generalization ability of estimators with the lowest possible computational cost. Many algorithms have been proposed in the literature to construct estimators with good generalization ability, especially those ones working with committee machines [2,3,4,5,6,7].

Committee machines is a research area where a combination of estimators is used for prediction instead of using a single model. The area of committee machines may be divided into two approaches: *ensemble* and *modular architecture*. The former combines redundant predictors in the sense that each one could solve the task as a whole [8], however, the best result is expected to be achieved by using the combination. In the modular approach, the problem is divided into different sub-tasks and each predictor takes charge of a sub-task whereas the final

solution has to be composed of all predictors [9]. In this case, the divide-and-conquer principle is applied and a complex problem may be transformed into a group of simpler tasks [10]. As a general rule, combining predictors is more robust and accurate than a single one [11]. A great number of algorithms has been proposed in the literature to automatically construct committee machines. The use of evolutionary computation to construct accurate committees has been explored and has shown important results [12,13,14,15,16].

One interesting approach to construct ensembles is the use of Artificial Immune Systems (AIS); computational techniques that mimic the immune system of species. The interest and applications of these techniques in a variety of research areas have grown in the last few years. Although just a few published papers have used AIS in the construction of ensembles [16,17,18,19], it seems to be a promising application of such techniques.

This work aims at constructing ensembles by means of Artificial Immune Systems and more precisely the Clonal Selection Algorithm (CSA) [20]. The difference with respect to previous AIS ensemble implementations [17,18] is that we implemented the selection and generation of the ensemble members as interdependent (parallel) tasks. In addition to that, the fitness of individuals is coupled with their performance during ensemble training (not to their individual performance as was implemented in [16,19]). Besides, the proposed implementation was tested in regression problems that have not attracted as much attention as classification problems in the ensemble literature. This can be seen as a first step towards solving identification problems by means of ensemble models.

This paper is organized as follows: In Sect. II, a brief introduction to ensembles is presented as well as the Clonal Selection Algorithm. The proposed algorithm is presented in Sect. III and Sect. IV presents the results on regression problems. Finally, conclusions are drawn in Sect. V.

2 Related Work

Krogh and Vedelsby [25] showed that, in a regression context, the squared error of the ensemble is lesser or equal than the mean squared error of the components, namely the ambiguity decomposition:

$$(f_{\text{ens}} - \Phi)^2 = \sum_i w_i (f_i - \Phi)^2 - \sum_i w_i (f_i - f_{\text{ens}})^2, \quad (1)$$

where $f_{\text{ens}} = \sum_i w_i f_i$ is the weighted combination of estimators f_i and Φ is the desired output. The first term of the right side is the weighted mean of the generalization error of each member and the second term is the variability among the members, namely ambiguity term (diversity). Thus, since the ambiguity term is always positive, the generalization error of the ensemble is lesser than the weighted mean generalization error of its members. In this way, it can be inferred that one important feature to be explored in the construction of ensembles is the diversity [22,23,24]. Moreover, the generalization error of the ensemble depends

on the correlation between its components in addition to the bias and variance of each member [21].

However, the search for diverse members may result in individuals with poor generalization. So, there is an optimum balance between these terms [26]. Although some authors have argued that the explicit use of diversity measure during the learning process to generate ensemble members may not be useful [27,28], there is a consensus that it is necessary to have diversity among the components to build an ensemble. Furthermore, diversity can be explicitly or implicitly produced.

Based on how diversity is created, [24] presented a new categorization of ensemble methods: (i) starting point in hypothesis space: for example varying the initial weights of a network; (ii) set of accessible hypothesis: changing the input training data of each member as the techniques *Bagging* [29] and *Boosting* [30] or manipulating the architecture of the components; (iii) hypothesis space traversal: which includes penalty methods [22] and evolutionary approaches.

As far as evolutionary computation is concerned, it has been successfully used to build ensembles [7]. An important field of evolutionary systems is Artificial Immune Systems that have been successfully applied in a variety of problems [31,32]. One interesting application of those systems is in ensemble learning.

Unlike other evolutionary algorithms, AIS can deliver not only one single solution (global optimum) but as well sub-optimum solutions (local optima) explicitly kept and evolved along the optimization process [20] which is a useful feature to construct ensembles. More details about AIS can be found in [32,33].

In this paper, the Clonal Selection Algorithm (CSA) [20] (Alg. 1) was implemented to evolve neural networks to compose the ensemble. In the CSA, the antibodies (potential solutions) that present higher affinity with an antigen (problem) have greater proliferation rates (number of clones). The cloned antibodies need to go through a mutation process, called hypermutation. The clones related to the best antibodies mutate into nearer new antibodies (performing a local search) while clones of the useless antibodies mutate into farther new antibodies (performing a global search).

3 Algorithm

The algorithm has just one objective-function to be minimized, the ensemble training error:

$$J = (f_{\text{ens}} - \Psi)^2, \quad (2)$$

where f_{ens} , output of the ensemble, is given by $\frac{1}{M} \sum_{i=1}^M f_i$ – where M is the ensemble size and f_i is the output of the ensemble member i – and Ψ is the training data target.

The members of the ensemble are supplied by the population of NN. Note that only part of the population is used to build the ensemble as was suggested in [34]. Although any kind of learning machines can be chosen, in this work the individuals of this population are feed-forward multi-layer perceptron neural networks with one hidden layer. The maximum number of hidden nodes N_h

Algorithm 1. *Clonal Selection Algorithm*

```

1: Input: Population  $P$ , selection rate  $SR$ , clone rate  $CR$ , and mutation radius  $\gamma$ 
2: Evaluate individuals in  $P$ 
3: Rank individuals
4: for  $i = 1, \dots, \text{round}(SR \cdot \text{size}(P))$  do
5:   Calculate the number of clones ( $NC$ ) of the individual:
      $NC_{ind_{rank_i}} \leftarrow \text{round}\left(\frac{CR \cdot \text{size}(P)}{i}\right)$ 
6:    $fit_{ind} \leftarrow \left(1 - \frac{rank_{ind}}{\text{size}(P)-1}\right)$ 
7:   for  $j = 1, \dots, NC_{ind_{rank_i}}$  do
8:      $cl = \text{clone}(ind_{rank_i})$ 
9:      $\alpha = \gamma \cdot e^{-fit_{ind_{rank_i}}}$ 
10:    Mutate( $cl, \alpha$ )
11:    Evaluate( $cl$ )
12:    if  $cl$  is better than  $ind_{rank_i}$  then
13:       $ind_{rank_i} \leftarrow cl$ 
14:    end if
15:  end for
16: end for
17: for  $i = \text{round}(SR \cdot \text{size}(P))+1, \dots, \text{size}(P)$  do
18:   Randomly generate new individual
19: end for
20: Output: New population  $P$ 

```

is defined by the user, but each NN may have different number of activated connections or hidden nodes. Each connection of the NN has two parameters: one binary to indicate if the connection is active or not, and the other is a real value representing the weight on a synapse.

The evolution of this population is done by means of the Clonal Selection Algorithm. The main advantages of applying this algorithm here are:

- there is no crossover operator, thus the individuals are isolated from each other keeping some diversity among them. Moreover, individuals with different topologies can be evolved in the same population;
- the hypermutation operator mutates the individuals regarding their fitness. The best individuals will have small mutation, searching for solutions around and close to them (exploitation), and the worst individuals will be more severely mutated (exploration);
- the best individuals are rewarded with more clones, that will help, for instance, the NN individuals that participate of the ensemble to find better results, as a local search, by means of the hypermutation operator;
- AIS provide more than one solution for a problem (they are able to store sub-optimum solutions together with the global optimum [20]), that is useful in the construction of ensemble since we are not looking for just one solution.

During initialization, a random population of NN is generated and a subset of individuals is randomly selected to form the ensemble (the subset size is defined

as the desired ensemble size). To evaluate the individuals of the population, one member of the ensemble (subset) is randomly selected (only if the individual being evaluated is not an ensemble member, i. e. it does not belong to the subset). Thus, the selected member is replaced by the NN individual being evaluated and the evaluation is calculated by (2). If the individual being evaluated achieves better performance than the replaced one, it will be a new ensemble member and all members of the ensemble (subset) will receive the same evaluation.

After evaluation, individuals are ranked (individuals of the ensemble receive the same rank, i. e. 0) and the number of clones and the *fitness* are calculated as presented in the Alg. 1. Then Clonal operators (generation of clones, mutation and evaluation) are applied to a pre-defined number of individuals.

There are two mutation operators for the NN: parametric and structural mutation. Firstly, structural mutation is applied (Alg. 2). Using the Clonal hypermutation parameter, $\alpha = \gamma \cdot e^{-fitness}$, where γ (Clonal mutation radius) is defined by the user, the inputs to hidden nodes weights and the bias weights are either de-activated (by means of a binary value) if a random number is lesser than α . For outputs of the hidden layer weights, the threshold of mutation is defined as $\alpha \cdot e^{-fitness}$ – since this threshold defines if a hidden node will be active or not, it was decreased to avoid big changes in the neural network structure. After structural mutation, parametric mutation is applied. In this way, part of the training data is randomly selected and the gradient descent backpropagation algorithm is applied with number of epochs defined by $round(e^{-2fitness} \cdot mutation_{epochs})$, where $mutation_{epochs}$ is defined by the user.

Algorithm 2. *Mutation of NN*

```

1: Input: individual NN, parameter  $\alpha$  (see Alg. 1)
2: {Structural mutation:}
3: for  $i = 1, \dots$ , number of binary parameters of individual do
4:   if parameter is related to input to hidden node connection or bias connections
     then
5:     if Random  $< \alpha$  then
6:       Change binary value
7:     end if
8:   else
9:     {Hidden nodes to output connections - activate or not hidden node}
10:    if Random  $< \alpha \cdot e^{-fitness_{individual}}$  then
11:      Change binary value
12:    end if
13:  end if
14: end for
15: {Parametric mutation:}
16: Apply backpropagation algorithm in the individual with number of epochs equals to
     $round(e^{-2fitness_{individual}} \cdot mutation_{epochs})$ , on part of the training algorithm randomly
    chosen
17: Output: Mutated NN individual

```

4 Experiments

The proposed algorithm, namely CLONENS (*CLONal Selection Algorithm for building ENsembles*), was tested on three benchmark synthetic regression problems used in [35] on multivariate regression splines (MARS). These problems are presented in Table 1. We generated 200 noisy data for the training process and 1000 testing data without noise to validate the models. All input data were uniformly and randomly generated using the domains shown in Table 1. The added noise of the first function, ϵ_1 , was Gaussian $N(0,1)$. The others, ϵ_2 and ϵ_3 , were adjusted to keep a signal to noise ratio of 3, as suggested by [35]. The first function has 10 independent random inputs (although just 5 have prediction power) and the other two functions have 4 input variables. All training data sets were normalized into the interval $[0,1]$.

The following parameters were fixed to all experiments: ensemble size (10), NN population size (200), number of generations (200), selection rate (0.9), clone rate (0.7), mutation radius (0.4), maximum number of backpropagation mutation epochs (100), number of hidden layers equals to 1, 5 hidden nodes and logsig activation function for all nodes. These parameters are not optimized.

To obtain first insights into the stability of the algorithm, only one training data set and one test data set were used to test the algorithm. We run the algorithm for 25 times with different initial seeds for the three functions. Table 2 shows the results of the proposed algorithm and demonstrate the stability of the algorithm through the low variance in the ensemble's performance.

Using the same data set of the previous experiment, feed-forward neural networks were trained using the Levenberg-Marquardt algorithm to compare with our approach. We used early stopping where the validation data was noise free. In addition to that, there were generalization error thresholds (3.5, 14000 and 0.1 for each function respectively) for the trained NN to be considered as an acceptable run. At the end of this process, 25 trained NN of 5 hidden nodes and tansig activation functions (best results were found with this configuration) were obtained. Table 3 presents the results of the early stopping approach. Comparing Tables 2 and 3, one remarkable result of CLONENS in relation to the early stopping procedure is its low variance. Besides, better generalization performance was obtained even though the early stopping approach has used noise-free data during validation and was subjected to a generalization error threshold.

For comparing CLONENS results with others presented in the literature, we run it 25 times for different randomly selected training sets. The early stopping

Table 1. Synthetic functions presented by [35] and used to test the proposed algorithm

Function	Domain
$f_1(x) = 10 \sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \epsilon_1$	$x_i \in [0,1], i=1 \dots 10$
$f_2(x) = \left(x_1^2 + \left(x_2 x_3 - \frac{1}{x_2 x_4} \right)^2 \right)^{\frac{1}{2}} + \epsilon_2$	$x_1 \in [0,100], x_2 \in [40\pi, 560\pi], x_3 \in [0,1], x_4 \in [1,11]$
$f_3(x) = \arctan\left(\frac{x_2 x_3 - (1/x_2 x_4)}{x_1}\right) + \epsilon_3$	$x_1 \in [0,100], x_2 \in [40\pi, 560\pi], x_3 \in [0,1], x_4 \in [1,11]$

Table 2. Results obtained by 25 runs of the proposed algorithm in one training data set (MSE). Individual and ensemble performances.

		Training				Generalization			
		Mean	SD	Best	Worst	Mean	SD	Best	Worst
f_1	Ind.	2.0563	0.6396	1.2909	4.5747	2.3142	0.8500	1.2348	6.2455
	Ens.	1.1505	0.0363	1.1008	1.2160	1.1161	0.0711	1.0022	1.2547
f_2	Ind.	42734	2762	39050	58254	7058	3760	2638	31296
	Ens.	38922	149.7	38613	39189	2673	135.8	2379	2939
f_3	Ind.	0.2359	0.0808	0.1986	2.2437	0.0744	0.0860	0.0224	2.0707
	Ens.	0.1953	0.0021	0.1929	0.2008	0.0326	0.0022	0.0279	0.0361

Table 3. Results of the early stopping procedure using the test data set as validation set, in the case of one training data set – 25 runs (MSE)

		Training				Generalization			
		Mean	SD	Best	Worst	Mean	SD	Best	Worst
f_1		1.1772	0.2979	0.9301	1.9925	1.4167	0.5873	0.7068	2.8042
f_2		39022	1570	35076	42295	6048	2353	3553	12896
f_3		0.2075	0.0108	0.1872	0.2328	0.0408	0.0086	0.0283	0.0627

algorithm and *bagging* technique were used to find 25 neural networks (or ensembles in the case of *bagging*) using the same training sets used by CLONENS algorithm (the validation set of 1000 noise free samples was the same in all experiments in this paper). The comparison among the proposed ensemble construction approach and other ensemble methods must be looked at with care because different procedures and data sets were used in the available papers. Putting these differences that are hard to quantify or find aside, it can be concluded that the proposed algorithm is very competitive.

In order to understand the implemented algorithm, the training error during the evolutionary process is visualized for each regression problem along with changes in the population and in the ensemble members. Because of space constraints, we only show the results for the third function in Figure 1. As can be inferred, during the first generations the number of times that an ensemble

Table 4. Comparison among generalization error (MSE) of different methods. The first column presents our proposed approach results, the second shows the early stopping (ES) procedure results and the third presents the implemented bagging algorithm results over 25 runs using 25 randomly generated data sets. Some results of the literature are shown in the other columns.

	CLONENS	ES	Bagg.	Bagg.[29]	Bagg.[36]	Bagg.[37]	Boost.[37]
f_1	1.0380	1.6069	2.8922	6.02	2.922	2.26	1.74
f_2	3943	6969	6753	21700	18448	10093	10446
f_3	0.0225	0.0482	0.0319	0.0249	0.0196	0.0303	0.0206

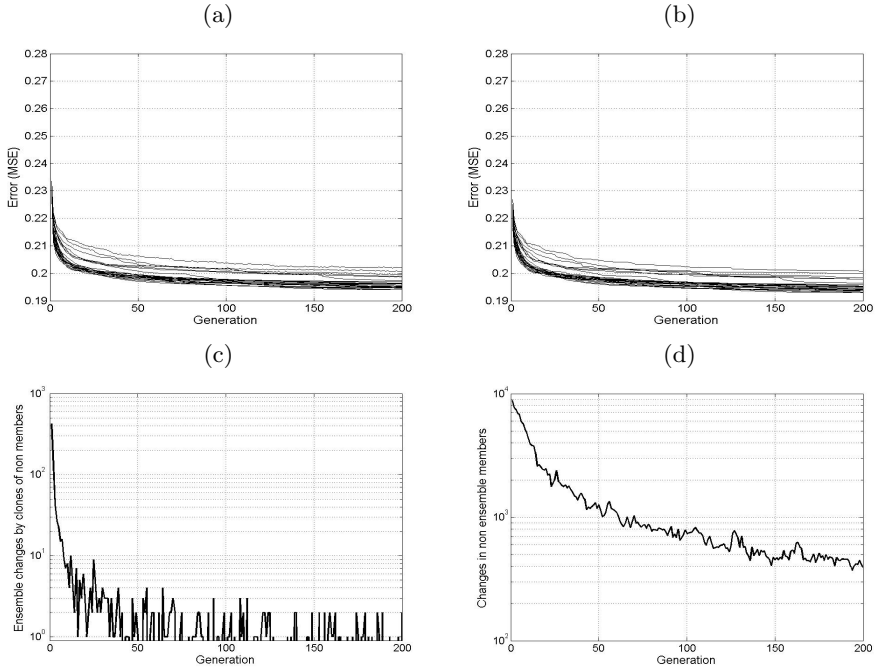


Fig. 1. Ensemble training error over generation plots for the 25 runs (Table 2, f_3). (a) mean error of the ensembles, (b) the best ensembles. Number of changes in: (c) different individual inserted in the ensemble replacing one ensemble member, (d) changes in the non member individuals. Values represent the sum of the 25 runs.

member was replaced by a non member individual was bigger suggesting an exploration behavior of the algorithm. After this period the main causes of changes in the ensemble were due to clones of ensemble members (exploitation). Nevertheless, not as frequent as before, ensemble members were still being replaced by non members during the evolutionary process.

5 Conclusion

In this paper, we introduced a novel ensemble construction approach. A population of NN was evolved by means of the Clonal Selection Algorithm. One important feature of the proposed algorithm is that the creation and selection of the ensemble members are done in the same evolutionary generation. Moreover, it allows the presence of individuals of different structures and different learning algorithms. In this way, the ensemble diversity is implicitly created. It was shown that the proposed algorithm achieved good generalization results. Although the use of Artificial Immune Systems is still not common in the ensemble literature, it seems to be very promising.

Future studies will examine the relationship among the parameters of the algorithm, more regression functions and a more detailed comparison with other

ensemble learning algorithms. Besides, different estimators could be used instead of neural networks and application of this approach in classification problems and identification of non-linear dynamical systems is also aimed. Diversity measures could also be applied during the evolution of the population.

Acknowledgments. This work has been supported by grants from CNPq, the National Council for Scientific and Technological Development - Brazil.

References

1. Mitchell, T.M.: Machine Learning. McGraw Hill, New York (1997)
2. Wolpert, D.H.: Stacked generalization. *Neural Networks* 5(2), 241–259 (1992)
3. Drucker, H., Cortes, C., Jackel, L.D., LeCun, Y., Vapnik, V.: Boosting and other ensemble methods. *Neural Computation* 6(6), 1289–1301 (1994)
4. Sollich, P., Krogh, A.: Learning with ensembles: How overfitting can be useful. In: Touretzky, D.S., Mozer, M.C., Hasselmo, M.E. (eds.) *Advances in Neural Information Processing Systems*, vol. 8, pp. 190–196. MIT Press, Cambridge (1996)
5. Opitz, D., Maclin, R.: Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research* 11, 169–198 (1999)
6. Dietterich, T.G.: Ensemble learning. In: Arbib, M.A. (ed.) *The Handbook of Brain Theory and Neural Networks*, 2nd edn., pp. 405–408. MIT Press, Cambridge (2002)
7. Yao, X., Islam, M.M.: Evolving artificial neural network ensembles. *IEEE Computational Intelligence Magazine* 3(1), 31–42 (2008)
8. Hansen, L.K., Salamon, P.: Neural networks ensembles. *IEEE Tran. Patterns Anal. Machine Intelligence* 12(10), 993–1001 (1990)
9. Sharkey, A.: Multi-Net Systems. In: *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems*, pp. 1–30. Springer, Heidelberg (1999)
10. Jacobs, R.A., Jordan, M.I., Nowlan, S.J., Hinton, G.E.: Adaptive mixtures of local experts. *Neural Computation* 3, 79–87 (1991)
11. Perrone, M.P., Cooper, L.N.: When networks disagree: ensemble methods for hybrid neural network. In: Mammone, R.J. (ed.) *Neural Networks for Speech and Image Processing*, pp. 126–142. Chapman Hall, Boca Raton (1993)
12. Abbass, H.A.: Pareto neuro-evolution: Constructing ensemble of neural networks using multi-objective optimization. In: *The 2003 Congress on Evolutionary Computation*, vol. 8(12), pp. 2074–2080 (2003)
13. Jin, Y., Okabe, T., Sendhoff, B.: Neural network regularization and ensembling using multi-objective evolutionary algorithms. *Congress on Evolutionary Computation* 1, 1–8 (2004)
14. Chandra, A., Yao, X.: Evolving hybrid ensembles of learning machines for better generalisation. *Neurocomputing* 69(7–9), 686–700 (2006)
15. Nguyen, M.H., Abbass, H.A., McKay, R.I.: A novel mixture of experts model based on cooperative coevolution. *Neurocomputing* 70, 155–163 (2006)
16. García-Pedrajas, N., Fyfe, C.: Construction of classifier ensembles by means of artificial immune systems. *Journal of Heuristics* 14(3), 285–310 (2008)
17. Castro, P.D., Coelho, G.P., Caetano, M.F., Zuben, F.J.V.: Designing ensembles of fuzzy classification systems: an immune approach. In: Jacob, C., Pilat, M.L., Bentley, P.J., Timmis, J.I. (eds.) *ICARIS 2005. LNCS*, vol. 3627, pp. 469–482. Springer, Heidelberg (2005)

18. Zhang, X., Wang, S., Shan, T., Jiao, L.: Selective SVMs ensemble driven by immune clonal algorithm. In: Rothlauf, F., Branke, J., Cagnoni, S., Corne, D.W., Drechsler, R., Jin, Y., Machado, P., Marchiori, E., Romero, J., Smith, G.D., Squillero, G. (eds.) *EvoWorkshops 2005*. LNCS, vol. 3449, pp. 325–333. Springer, Heidelberg (2005)
19. García-Pedrajas, N., Fyfe, C.: Immune network based ensembles. *Neurocomputing* 70(7-9), 1155–1166 (2007)
20. de Castro, L.N., Zuben, F.J.V.: Learning and optimization using the clonal selection principle. *IEEE Trans. on Evolutionary Computation* 6(3), 239–251 (2002)
21. Ueda, N., Nakano, R.: Generalization error of ensemble estimators. In: *IEEE International Conference on Neural Networks*, vol. 1, pp. 90–95 (1996)
22. Liu, Y., Yao, X.: Ensemble learning via negative correlation. *Neural Networks* 12(10), 1399–1404 (1999)
23. McKay, R., Abbass, H.A.: Anti-correlation: a diversity promotion mechanisms in ensemble learning. *The Australian Journal of Intelligent Information Processing Systems* 7(3), 139–149 (2001)
24. Brown, G., Wyatt, J., Harris, R., Yao, X.: Diversity creation methods: A survey and categorisation. *Journal of Information Fusion* 6(1), 5–20 (2005)
25. Krogh, A., Vedelsby, J.: Neural network ensembles, cross validation, and active learning. In: Tesauro, G., Touretzky, D.S., Leen, T.K. (eds.) *Advances in Neural Information Processing Systems* 7, pp. 231–238. MIT Press, Cambridge (1995)
26. Brown, G., Wyatt, J., Tino, P.: Managing diversity in regression ensembles. *Journal of Machine Learning Research* 6, 1621–1650 (2005)
27. García-Pedrajas, N., Hervás-Martínez, C., Ortiz-Boyer, D.: Cooperative coevolution of artificial neural network ensembles for pattern classification. *IEEE Transactions on Evolutionary Computation* 9(3), 271–302 (2005)
28. Kuncheva, L.I., Whitaker, C.J.: Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning* 51(2), 181–207 (2003)
29. Breiman, L.: Bagging predictors. *Machine Learning* 24(2), 123–140 (1996)
30. Schapire, R.E.: The strength of weak learnability. *Machine Learning* 5(2), 197–227 (1990)
31. Dasgupta, D., Ji, Z., González, F.: Artificial immune system (AIS) research in the last five years. In: *IEEE International Conference on Evolutionary Computation*, Canberra, Australia (2003)
32. de Castro, L.N., Timmis, J.: *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer, London (2002)
33. Dasgupta, D. (ed.): *Artificial Immune Systems and Their Applications*. Springer, Heidelberg (1998)
34. Zhou, Z.H., Wu, J., Tang, W.: Ensembling neural networks: many could be better than all. *Artificial Intelligence* 137(1-2), 239–263 (2002)
35. Friedman, J.H.: Multivariate adaptive regression splines. *The Annals of Statistics* 19, 1–141 (1991)
36. Borra, S., Ciaccio, A.D.: Improving nonparametric regression methods by bagging and boosting. *Comput. Stat. Data Anal.* 38(4), 407–420 (2002)
37. Drucker, H.: Improving regressors using boosting techniques. In: *ICML 1997: Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 107–115. Morgan Kaufmann Publishers Inc., San Francisco (1997)

Improving the Performance and Scalability of Differential Evolution

Antony W. Iorio and Xiaodong Li

School of Computer Science and Information Technology,
RMIT University,
Melbourne Australia
iantony@gmail.com,
xiaodong@goanna.cs.rmit.edu.au
<http://goanna.cs.rmit.edu.au/~xiaodong/ecml>

Abstract. Differential Evolution (DE) is a powerful optimization procedure that self-adapts to the search space, although DE lacks diversity and sufficient bias in the mutation step to make efficient progress on non-separable problems. We present an enhancement to Differential Evolution that introduces greater diversity. The new DE approach demonstrates fast convergence towards the global optimum and is highly scalable in the decision space.

Keywords: Differential Evolution, Optimization, Rotational Invariance.

1 Introduction

Despite the power of many population-based stochastic optimization algorithms, they can meet with difficulties on optimization problems which are non-separable. Traditional Genetic Algorithms fail to optimize these problems efficiently because they typically perform independent perturbations of decision variables. Unfortunately, many real-world problems are not linearly separable. On problems which are not aligned with the principle coordinate axes, the small mutation rates frequently used in Genetic Algorithms are known to be even less efficient than a random search [1]. One approach for optimizing such problems is to use a vector-based scheme such as Differential Evolution.

The Differential Evolution (DE) optimization algorithm works by generating difference vectors between points in the search space, and using the resulting scaled difference vector to perturb existing points in the population [2]. For example offspring can be generated using the DE scheme $\mathbf{x}^{(i)} + F(\mathbf{x}^{(r1)} - \mathbf{x}^{(r2)})$ where $r1$ and $r2$ refer to the indices of two distinct randomly selected individuals from a population, and i is the index of the current individual in the population. In this approach, individuals are selected for a DE operation such that the resulting difference vector, $(\mathbf{x}^{(r1)} - \mathbf{x}^{(r2)})$, has a magnitude greater than 0. The addition of $(\mathbf{x}^{(r1)} - \mathbf{x}^{(r2)})$ to $\mathbf{x}^{(i)}$ is considered to be a mutation operation which perturbs $\mathbf{x}^{(i)}$. F is a scaling factor for the difference vector.

Differential Evolution has a number of attractive features; difference vectors can be correlated with the search space, it uses only $\mathcal{O}(Np)$ processes (where Np is the population size), it doesn't need a predefined probability distribution for generating offspring, the objective functions do not need to be differentiable, it can provide multiple solutions from a single run of the algorithm, it is very simple to implement, and is a parallel optimization procedure like many other population based schemes.

Of course DE has some limitations, which we attempt to address in this paper. It has been reported that DE performs poorly on problems that are not linearly separable because of inefficient exploitation during the differential mutation phase [3]. Two hypotheses were explored by Sutton; when the crossover rate (Cr) is low, DE can exploit the separability of a function. When DE has a Cr of 1.0, DE becomes rotationally invariant and depends entirely on the differential mutation step. In order to efficiently solve non-separable problems, DE typically must depend more on mutation than crossover, although DE lacks selection pressure in the differential mutation step to make efficient progress. Furthermore, making Cr equal to 1.0 is not recommended as it reduces the number of trial vectors and can result in stagnation [4]. Finally, DE becomes highly dependent on population size in order to avoid stagnation when no crossover is employed.

2 Problem Statement

The use of crossover in DE introduces diversity to the population, far more than mutation alone. As the decision space dimension scales, the importance of having a diverse population from which to sample becomes significantly important in order to make efficient progress towards more optimal solutions in the search space. Unfortunately, because the offspring that crossover can generate are dependent on the principle coordinate axes, crossover provides little benefit to the optimization of non-separable problems.

For an algorithm to be rotationally invariant in the context of optimization algorithms it should produce offspring in the same relative location, irrespective of the orientation of the fitness landscape. Although the rotationally invariant DE/rand/1/bin approach provides vector wise samples which are not biased with respect to any particular coordinate axes it also lowers the number of potential offspring dramatically because it does not use crossover [4]. Our contention is that in order for an optimization algorithm to perform efficiently on a non-separable problem it must not exhibit an extreme dependency on the principle coordinate axes. In addition, we contend that it is unnecessary for it to be strictly rotationally invariant, as long as it is capable of generating sufficient diversity.

It is important to elucidate further on one of the reasons for why the rotationally invariant DE/rand/1/bin used in this study performs poorly on non-separable problems. In Figure 1 the offspring and parents are represented for a population size of 4. The number of potential unique offspring that can be sampled for a single base-vector by such a scheme is determined by Equation 1,

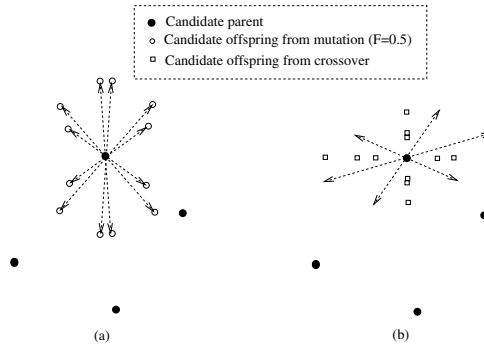


Fig. 1. (a) The offspring generated from a population of 4 using the CSDE scheme. (b) The offspring generated from a population of 4 using the basic DE/rand/1/bin scheme *with* and *without* crossover.

$$(Np - 1)(Np - 2)(2^D - 1) \quad (1)$$

where D is the decision space dimension. The term 2^D represents the number of possible offspring that can be generated from binomial crossover. The term $(Np - 1)(Np - 2)$ is the number of possible offspring that can be generated from vector-wise mutation. Also, crossover can produce duplicate individuals that were already sampled. In order to not count these individuals, we subtract the duplicates. It deserves to be noted that Equation 1 is equivalent to previous results which reported upon the number of samples possible in an entire population [4].

In accordance with this equation and the offspring distributions in Figure 1(b), an enumeration of all 18 possible offspring for a single target vector is provided in Table 1. The first column of this table details all possible differentials from Figure 1(b). The second contains the location of offspring produced by a mutation operation with $F=0.5$. The third column contains the coordinates of unique offspring resulting from crossover, which do not overlap with offspring resulting from a mutation operation or any existing parents. The total number of offspring possible from a population of 4 individuals where $0 < CR < 1.0$ is 18.

It is apparent from Equation 1 that as the decision space dimension scales, crossover is responsible for the majority of the offspring individuals that the algorithm can generate through the 2^D term [4]. It is also clear from this figure that crossover samples along the principle coordinate axes, so although it generates many offspring, it also constrains them to this region. It is only capable of independent sampling in each decision space dimension.

If we consider Figure 1(b), where crossover is absent, but rotational invariance is maintained, significantly fewer offspring can be sampled for a single base-vector. The number of potential offspring that can be sampled is equal to $(Np - 1)(Np - 2)$. The implication here is that a rotationally invariant DE scheme is highly dependent on the population size in order to maintain sample diversity. Although it samples offspring independent of any particular coordinate axes, it does not scale in the decision space as well as a scheme incorporating crossover.

Table 1. An enumeration of all 18 possible unique offspring for a single target vector resulting from crossover and mutation operations in Figure 1(b). The base-vector, $\mathbf{x}^{(i)}$ is located at (2.0,2.5). In addition, the coordinates of A, B, and C are respectively (0.0,0.0), (2.3,-1.0) and (4.0,1.0).

$(\mathbf{x}^{(r1)} - \mathbf{x}^{(r2)})$	$\mathbf{x}^{(i)} + F(\mathbf{x}^{(r1)} - \mathbf{x}^{(r2)})$	Crossed with $\mathbf{x}^{(i)}$
A-B = (-2.3,1.0)	(1.15,3.0)	$\{(1.15,2.5),(2.0,3.0)\}$
B-A = (2.3,1.0)	(3.15,2.0)	$\{(3.15,2.5),(2.0,2.0)\}$
A-C = (-4.0,-1.0)	(0.0,2.0)	$\{(0.0,2.5),(2.0,2.0)\}$
C-A = (4.0,1.0)	(4.0,3.0)	$\{(4.0,2.5),(2.0,3.0)\}$
B-C = (-1.7,-2.0)	(1.85,1.5)	$\{(1.85,2.5),(2.0,1.5)\}$
C-B = (1.7,2.0)	(2.85,3.5)	$\{(2.85,2.5),(2.0,3.5)\}$

Ideally, we would like a scheme which is biased in order to accelerate convergence, is capable of generating a diverse variety of offspring solutions in a manner which minimizes distribution bias, and is capable of optimizing non-separable and separable problems equally well. Furthermore, it should be simple to implement and computationally efficient.

As we mentioned earlier, traditional crossover offers sampling diversity, but is really only effective on separable problems because of the way it generates points. It would be desirable for DE to have the capability of producing a large number of samples, while still remaining effective on non-separable problems in high decision space dimensions.

3 An Improved Sample-Based DE Algorithm

In this section we describe the Combinatorial Sampling Differential Evolution (CSDE) algorithm which uses a ‘target’ best individual and maintains diversity using the sampling of difference vectors from two parent vectors. For the purpose of simplicity we describe the behavior of the algorithm in a two dimensional decision space, although the process easily generalizes to an arbitrary number of decision space dimensions. We also discuss some of the advantages and characteristics of the approach which are different from the typical DE.

Firstly, the algorithm loops over all individuals in a population such that each individual $\mathbf{x}^{(i)}$ has an opportunity to participate in the DE calculation. A second individual $\mathbf{x}^{(r)}$ is chosen for a difference vector calculation such that the population index r is not equal to i , and r is an index randomly chosen from the population.

Two types of samples are performed in this algorithm around an individual that is deemed to be better than another. The first type of sample we call a C-sample (*correlated sample*) such that the vector difference and perturbation are in the same direction, around a better individual (In Figure 2, $\mathbf{x}^{(i)}$ is better than $\mathbf{x}^{(r)}$, for the purpose of explaining the operation of the algorithm. Of course, if the opposite was true, then sampling would occur around individual $\mathbf{x}^{(r)}$). The point labeled by ①② corresponds to the point specified by Equation (2) and (3).

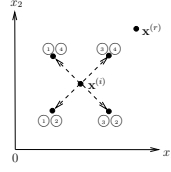


Fig. 2. In a 2-dimensional decision space vectors are sampled around a ‘better’ individual

In these equations, $u_1^{(i)}$ represents the offspring parameter from the DE mutation equation for the first parameter in the decision vector, and $u_2^{(i)}$ represents the offspring for the second parameter in the decision vector. Similarly, the point labeled by $\textcircled{3}\textcircled{4}$ corresponds to the point specified by Equation (4) and (5). Both points $\textcircled{1}\textcircled{2}$ and $\textcircled{3}\textcircled{4}$ are correlated because they are in the same direction as the difference vector. The points at $\textcircled{1}\textcircled{2}$ and $\textcircled{3}\textcircled{4}$ are sampled with the same probability.

$$u_1^{(i)} = x_1^{(i)} + F(x_1^{(i)} - x_1^{(r)}) \quad (2)$$

$$u_2^{(i)} = x_2^{(i)} + F(x_2^{(i)} - x_2^{(r)}) \quad (3)$$

$$u_1^{(i)} = x_1^{(i)} + F(x_1^{(r)} - x_1^{(i)}) \quad (4)$$

$$u_2^{(i)} = x_2^{(i)} + F(x_2^{(r)} - x_2^{(i)}) \quad (5)$$

The second type of sample is labeled in Figure 2 by $\textcircled{1}\textcircled{4}$ and $\textcircled{3}\textcircled{2}$ which respectively correspond to the points generated by Equation 2 and 5 and Equation 4 and 3. Both of these samples are uncorrelated and not rotationally invariant because the magnitudes of the difference vectors for these samples is the result of the difference between $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(r)}$ and they vary depending on the orientation of $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(r)}$. It is this second type of sampling that contributes diversity to the search. In traditional DE, only a single difference vector can result from two points. Our approach dramatically increases the number of possible samples at the expense of always generating rotationally invariant correlated samples. We call these sample points UC-samples (*uncorrelated and correlated samples*), and if the algorithm samples such points, it does so with equal probability for each possible point, including the rotationally invariant correlated sample points $\textcircled{1}\textcircled{2}$ and $\textcircled{3}\textcircled{4}$ and the uncorrelated points $\textcircled{1}\textcircled{4}$ and $\textcircled{3}\textcircled{2}$. As the decision space dimension scales, the number of such samples increases in proportion to 2^D , where D is the decision space dimension. In two decision space dimensions, there are four equations that can specify the possible sample points. In three dimensions, there will be eight equations. This can easily be implemented programmatically by specifying an equal probability for $u_j^{(i)} = x_j^{(i)} + F(x_j^{(i)} - x_j^{(r)})$ and $u_j^{(i)} = x_j^{(i)} + F(x_j^{(r)} - x_j^{(i)})$ to be used for each decision space parameter $j = 1$ to D , so that all possible samples have an equal chance of occurring.

Whether a C-sample or UC-sample occurs is determined probabilistically by a control parameter κ . This parameter is responsible for controlling the balance between C-sampling and UC-sampling in the generation of offspring.

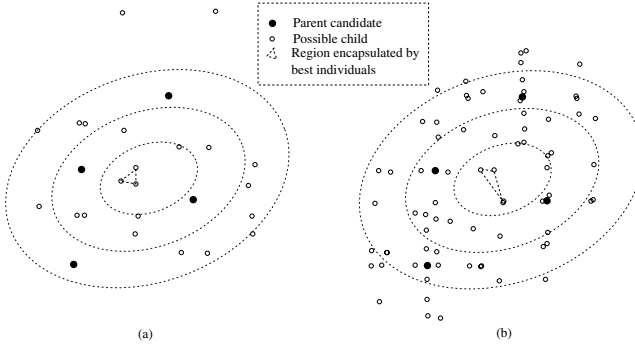


Fig. 3. Distribution of all possible offspring from 3 parent individuals and a mutation scaling factor $F=0.5$. (a) CSDE and (b) DE/rand/1/bin with crossover.

3.1 Characteristics and Advantages

In CSDE there are two pressures in the generation of offspring; exploitation results from the highly correlated rotationally invariant samples (C-samples) being generated, which rapidly drives the algorithm towards better solutions, and exploration occurs from the UC-sampling, which attempts to discover new and diverse points around the better individual. The UC-sampling method sacrifices emphasis on correlated rotationally invariant points for a dramatic increase in diversity as the decision space dimension scales to higher dimensions. The general idea of this approach is to increase the diversity that DE is capable of generating using a relatively small population size. A critical point to consider here is that although crossover is also not a rotationally invariant scheme, it only generates points which are aligned with the target parent. The UC-samples are not biased in such a fashion, and although they do not result in rotational invariance, they do produce offspring sample vectors distributed around the target vector.

An attractive feature of CSDE is that the number of potential offspring that can be sampled is bounded by 2^D , as in the crossover based DE described in Figure 1(b). Figure 1(a) shows how offspring are sampled using our approach. The number of candidates that can potentially be sampled around a base-vector in the sampling based approach is in proportion to $(Np - 1)2^D$. It is apparent from Figure 1(a) that CSDE is superior to standard DE with crossover because it can generate points that are not solely sampled along the principle coordinate axes, unlike DE with crossover in Figure 1(b). As a result, CSDE can be highly effective on problems which are non-separable compared with an algorithm which only produces biased samples along the principle coordinate axes aligned with a parent target vector. Furthermore, it bears mentioning that the smallest population size that CSDE can work with is 2, unlike standard DE which requires 4 individuals.

The difference in efficiency between the CSDE approach and DE/rand/1/bin is detailed further in Figure 3 where the CSDE is capable of generating offspring in near optimal regions of the search space with far fewer samples than the

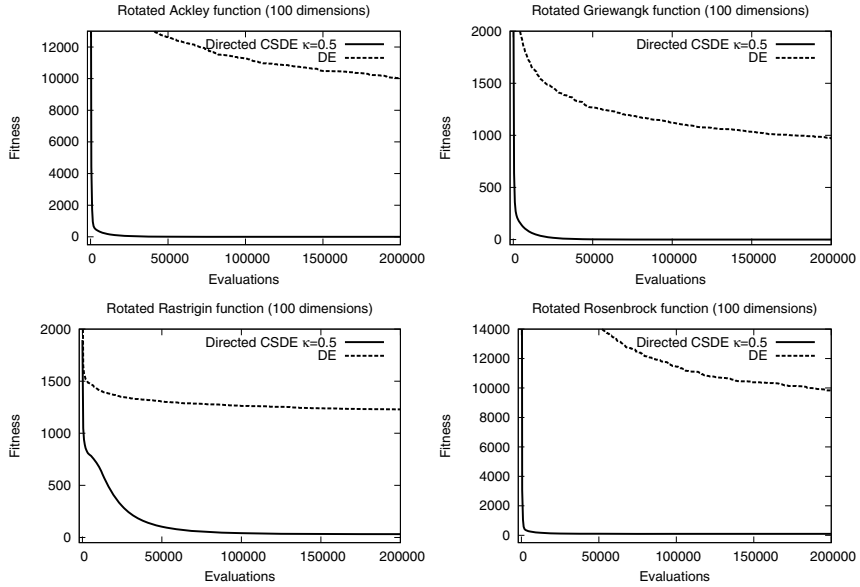


Fig. 4. Fitness over 200,000 evaluations on rotated problems in 100 dimensions of the decision space for the rotated Ackley, Griewangk, Rastrigin and Rosenbrock functions

DE/rand/1/bin approach. Although crossover based DE generates more points, it is apparent from this figure that the sampling based CSDE scheme is superior because of the greater focus it produces on more promising regions of the search space. In contrast, it is apparent that crossover based DE/rand/1/bin samples many more offspring but such offspring may not be advantageous towards finding the optima efficiently.

The proposed sampling based approach can be efficiently directed towards more optimal regions using appropriate vector selection. The CSDE approach is also capable of generating more points than standard rotationally invariant DE and like standard DE, only $\mathcal{O}(Np)$ processes are required.

4 Experiments and Methodology

In the CSDE approach described in this paper, an individual is deemed to be better than another individual with respect to fitness in order to determine an appropriate direction for the vector difference. If a better direction is not apparent from the measures of fitness associated with both individuals, then a direction is chosen randomly. In addition, if an individual has better fitness than another individual it replaces the inferior individual in the population.

Four DE variants are evaluated in this study. Firstly, a baseline DE technique incorporating three vectors is employed. This baseline approach was briefly discussed in the introductory section. For our purposes the baseline DE algorithm

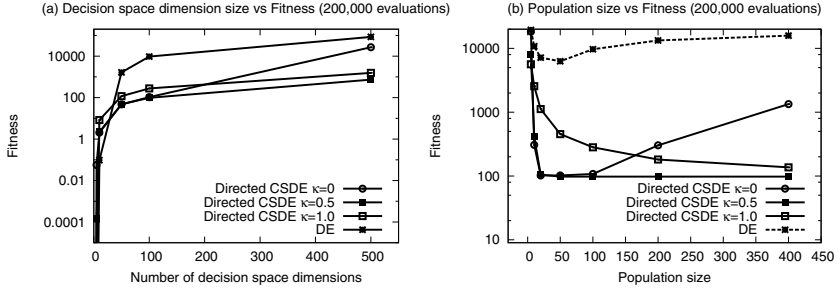


Fig. 5. Problem dimensionality vs. fitness and population size vs fitness after 200,000 evaluations on the rotated Rosenbrock function

used here for benchmarking is equivalent to the DE/rand/1/bin approach [4] because we use it with $CR = 1.0$ in this study.

Secondly, the CSDE algorithm is evaluated with κ set to 1.0, 0.5 and 0. When κ is set to 0.5, half the time the algorithm favors C-samples that are highly directed towards better solutions, otherwise it performs UC-sampling. When κ is set to 0 there is no bias, and vectors are sampled using UC-sampling only. In addition, when κ is set to 1.0, the algorithm solely performs C-samples.

A population size of 100 individuals is used for each of the algorithms on each of the test problems for the performance evaluation of the variants over time, with fitness on the vertical axis. For the DE variants, F is set to 0.5.

The problems that are used in the evaluation of each of the algorithms evaluated in this paper are the rotated Rosenbrock, Griewangk, Ackley and Rastrigin functions. In addition, the Rosenbrock function is evaluated with the algorithm variants in order to determine sensitivity to population size and scalability in the decision space.

A rotation matrix is used to introduced parameter interactions between decision variables, thereby making the problem non-separable. Rotations for each of these test problems are performed in the decision space, on each plane, using a random uniform rotation matrix, which introduces parameter interactions between all parameters [7]. Each algorithm is run 50 times on each test problem, for a total of 200,000 problem evaluations for each run. A new random uniform rotation matrix is generated for each run of each algorithm for the purpose of an unbiased assessment.

5 Scalability in the Decision Space and Sensitivity to Population Size

From Figure 5(a) it is apparent that CSDE with $\kappa = 0.5$ is insensitive to an increase in decision space size on the 100 dimensional rotated Rosenbrock function. In addition, it is able to find highly competitive solutions which are far superior to a canonical rotationally invariant DE/rand/1/bin approach, which performed poorly. In addition, in Figure 5(b) it is apparent that the CSDE

approach with $\kappa = 0.5$ is highly insensitive to a change in population size and is capable of finding similarly good solutions after 200,000 evaluations because of the order of magnitude higher degree of sampling that is possible compared with the canonical rotationally invariant DE/rand/1/bin. In contrast, the performance of CSDE with $\kappa = 0$ where only U-sampling is performed, peaks in performance between a population size of 20 to 100 individuals on the rotated Rosenbrock function in 100 dimensions. This indicates that a large population size detracts from the performance of CSDE when U-sampling is used. The reasons for this are that as the number of individuals in the population increases, the probability of sampling highly directed correlated samples reduces when U-sampling is employed. It is apparent that rotationally invariant correlated sampling is beneficial to the performance of the CSDE approach, in order to make it more insensitive to the population size. In contrast, the CSDE approach with $\kappa = 1.0$ is highly dependent on population size for sampling diversity, and the performance only begins to approach CSDE with $\kappa = 0.5$ as the population size approaches 500 individuals on this problem. It is also clear from Figure 5 that rotationally invariant DE/rand/1/bin performs extremely poorly on the 100 dimensional Rosenbrock problem in the presence of parameter interactions.

From these results it is clearly apparent that when $\kappa = 0.5$, the CSDE algorithm has superior performance over the UC-sampling approach which uses $\kappa = 0$. This indicates that sampling of highly directed rotationally invariant correlated vectors is critically important in order for the algorithm to remain insensitive to population size variations as well as discover highly fit solutions in extremely large decision spaces.

In order to test the performance of the variants over time, the rotated Ackley, Rastrigin, Griewangk and Rosenbrock functions were employed with 100 dimensions. The results in Figure 4 indicate that the performance of the CSDE approach with $\kappa = 0.5$ are dramatically superior to the rotationally invariant DE/rand/1/bin algorithm which does not employ sampling.

6 Implications and Conclusion

In this work we have addressed the stagnation issue discussed by [4] with the CSDE approach. Until now, in order to overcome stagnation in DE a very large population size had to be employed, or crossover was used in order to add more sampling diversity even though crossover is typically ineffective when optimization problems have many parameter interactions. Furthermore, rotationally invariant DE applied to non-separable problems is limited to rather low decision space dimensions and is highly dependent on population size. In contrast, the CSDE approach is insensitive to population size on the test problems used, even though it does not employ crossover in the traditional sense. It can also handle problems with parameter interactions in high dimensional spaces very well even though it is not a strict rotationally invariant algorithm. The results presented in this paper are significantly important to practitioners who are interested in

optimizing non-separable problems. Until now, previous work in this area focussed on computationally expensive Evolutionary Strategy techniques.

We have presented a computationally efficient, simple optimization algorithm for dramatically improving optimization performance on non-separable problems in high dimensional spaces. Although the comparative evaluation presented here was rather limited and only showed a comparison with the DE/rand/1/bin algorithm, the results are very promising. We intend to perform a more comprehensive study with a variety of DE algorithms and test problems in a future study.

References

1. Salomon, R.: Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions: A survey of some theoretical and practical aspects of genetic algorithms. *Bio. Systems* 39(3), 263–278 (1996)
2. Price, K.: Differential evolution: a fast and simple numerical optimizer. In: *Biennial Conference of the North American Fuzzy Information Processing Society*, New York, vol. 3339, pp. 524–527 (1996)
3. Sutton, A.M., Lunacek, M., Whitley, L.D.: Differential evolution and non-separability: using selective pressure to focus search. In: *GECCO 2007: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pp. 1428–1435 (2007)
4. Lampinen, J., Zelinka, I.: On stagnation of the differential evolution algorithm. In: *Proceedings of MENDEL 2000, 6th International Mendel Conference on Soft Computing*, pp. 76–83 (2000)
5. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* 6(2), 182–197 (2002)
6. Deb, K., Goyal, M.: A combined genetic adaptive search (GENEAS) for engineering design. *Computer Science and Informatics* 26(4), 30–45 (1995)
7. Iorio, A., Li, X.: Rotated test problems for assessing the performance of multi-objective optimization algorithms. In: *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pp. 683–690 (2006)
8. Zitzler, E., Thiele, L.: Multiobjective optimization using evolutionary algorithms - a comparative case study. In: *Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, pp. 292–304. Springer, Heidelberg (1998)*

A Fuzzy-GA Decision Support System for Enhancing Postponement Strategies in Supply Chain Management

Cassandra X.H. Tang and Henry C.W. Lau

Dept. of Industrial and Systems Engineering
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
cassandra.tang@polyu.edu.hk

Abstract. This paper aims to propose a knowledge-based Fuzzy - GA Decision Support System with performance metrics for better measuring postponement strategies. The Fuzzy - GA approach mainly consists of two stages: knowledge representation and knowledge assimilation. The relevant knowledge of deciding what type of postponement strategies to adopt is encoded as a string with a fuzzy rule set and the corresponding membership functions. The historical data on performance measures forming a combined string is used as the initial population for the knowledge assimilation stage afterwards. GA is then further incorporated to provide an optimal or nearly optimal fuzzy set and membership functions for related performance measures. The originality of this research is that the proposed system is equipped with the ability of assessing the loss caused by discrepancy away from the different supply chain parties, and therefore enabling the identification of the best set of decision variables.

Keywords: Supply chain management, Hybrid optimisation algorithms.

1 Introduction

Every company wishes to produce exactly what customers want at the proper place and time they wish to consume [1]. Supply chains need to be responsive to rapidly changing markets because of the mounting focus on customer satisfaction and product proliferation, which naturally magnify the complexity of demand forecasting and planning. Postponement turns to be one of the strategies to solve this problem.

Postponement is defined as the delay of the point of product differentiation in a production process to the latest possible time [2]. The company exerts control not through ownership, but rather by mastering the flow of information while focusing its competitive edge on the quality of its responsiveness to customers. Accordingly, the value of postponement is then regarded as the value of information since given that production decision time could be delayed, more latest information about the customer demand could be updated and analyzed more precisely [3].

Hence the quality of decision variables shall be optimized. Consequently, it improves the quality of the demand forecast as the forecasting point moves closer to production period [4-5]. It also allows flexibility in production scheduling to actual demand resulting in a more responsive supply chain network (SCN) [3].

The concept of postponement was first defined by Alderson [6] in marketing. Up till now its application and scope has expanded to areas such as manufacturing and distribution, and recently, to product and process re-design. Past classifications of postponement strategies concentrate mainly on manufacturing and distribution. With the growing emphasis on supply chain, there is an emerging need to re-consider the postponement strategies which encompass the whole supply chain. Besides, the lack of a comprehensive and exact performance measuring scheme especially for postponement is also observed. In response to that, the existing measurements [27-28] will be assessed by the proposed knowledge-based intelligent framework of performance metrics to advance the postponement strategies.

2 Literature Review

2.1 Postponement

The risk pooling concept is applied on the logistics postponement strategy by stocking differentiated products at the strategically central locations which can possibly balance between inventory cost and response time [7-8]. For instance, in the HP Deskjet printer case, HP decided to perform local customization in European countries for the printer line by postponing the final assembling procedure, that is, storing the semi-finished products in the local warehouse and carrying out the local customization process at the distribution centers in Europe [2]. This strategy successfully enabled the company to reduce the inventory level to some extent while maintaining or even raising the customer service level. Other examples such as Benetton Case [9] show the great achievement of postponement implementation.

With a high degree of conformance to the customers' assorted requirements, postponement has been identified as a crucial approach for contributing to the attainment of agility, e.g., through its contribution to the products and services customization, and a number of cross functional efforts [10-11].

Nowadays Christopher [12] even states that postponement is an essential element in any agile strategy of e-commerce, which implies that there are opportunities to re-engineer the existing supply chains or design new supply chains using postponement strategies [13-15].

2.2 Performance Metrics

Bowersox and Closs [16] stated three objectives for developing and implementing performance measurement systems: to monitor historical system performance for reporting, to control ongoing performance so that abnormal processes may be prevented, and to direct the personnel's activities. Although their discussion focused on logistics management, these three objectives are suitable to describe the motivation of postponement performance measurement due to several reasons: Firstly, the design and implementation of postponement affects the manufacturer's production and logistics function, consequently impacts the whole SCN. Secondly, some postponement strategies, such as place postponement, require the reconfiguration of the SCN. Thirdly, postponement strategy is essentially an information strategy carried throughout the SCN and can reduce uncertainty and improve forecast accuracy [3].

2.3 Fuzzy-GA Knowledge Processing Framework

Much work has been done in machine learning for classification; the ultimate goal is to attain a more accurate prediction. Artificial intelligence has been widely used in knowledge discovery by considering both cognitive and psychological factors. In some research development, GA, one of the search algorithms based on the mechanics of natural selection and natural genetics [17-18], has been regarded as a genetic optimization technique for global optimization, constrained optimization, combinatorial optimizations and multi-objective optimization. Recently, GA has been used to enhance industrial engineering for the achievement of high throughput and high quality [19-21]. In addition, one of the traditional fundamental concepts of fuzzy logics is that it is characterized by a qualitative, subjective nature and linguistically expressed values [22]. Studies on applying fuzzy logics to systems for different sectors have been widely undertaken.

Recently fuzzy rule sets, together with the associated membership functions, have been proven of great potential in their integration into GA to formulate a compound knowledge processing system [23]. On the other hand, trade-off on variable settings is also a crucial aspect in the manufacturing process. To achieve this goal, Leung and Spiring [24] have introduced the concept of the Inverted Beta Loss Function (IBLF), which is a further deduction of the Taguchi Loss Function [25] in the industrial domain, helping to balance the possible loss resulting from the use of different performance measure combinations along the entire process.

Inspiring from all above, a Fuzzy-GA Decision Support System is proposed for decision-makers to better select the proper postponement strategies in terms of the corresponding performance metrics.

3 The Proposed Fuzzy-GA Decision Support System

The proposed approach consists of two processes: knowledge representation and knowledge assimilation (Figure 1). In the first stage, the expertise of factor setting, which is represented by IF-THEN rules, is encoded as a string with fuzzy rule sets and the associated fuzzy membership function. The historical process data are also included into the strings mentioned above, contributing to the formulation of an initial knowledge population. Then in knowledge assimilation, GA is used to generate an optimal or nearly optimal fuzzy set and membership functions for the entitled performance indicators. The approach is to be demonstrated by the employment of an illustrative example, the trade-off between quality and overall financial performance. Accordingly, it is necessary to set relative weights for them to aggregate the measurement results since there naturally contains essential fuzziness and ambiguity in human judgments.

3.1 Problem Formulation

Fuzzy-GA is proposed for capturing domain knowledge from an enormous amount of data. The proposed approach is to represent the knowledge with a fuzzy rule set and encode those rules together with the associated membership into a chromosome.

A population of chromosomes comes from the past historical data and an individual chromosome represents the fuzzy rule and the related problem. A binary tournament, using roulette wheel selection, is used for picking out the best chromosome between 2 when a pair of chromosomes is drawn. The fitness value of each individual is calculated using the fitness function by considering the accuracy and the trade-off of the resulting performance measure setting, where the fitter one will remain in the population pool for further mating. After crossover and mutation, the offspring will be evaluated by the fitness function and the optimized solution will then be obtained.

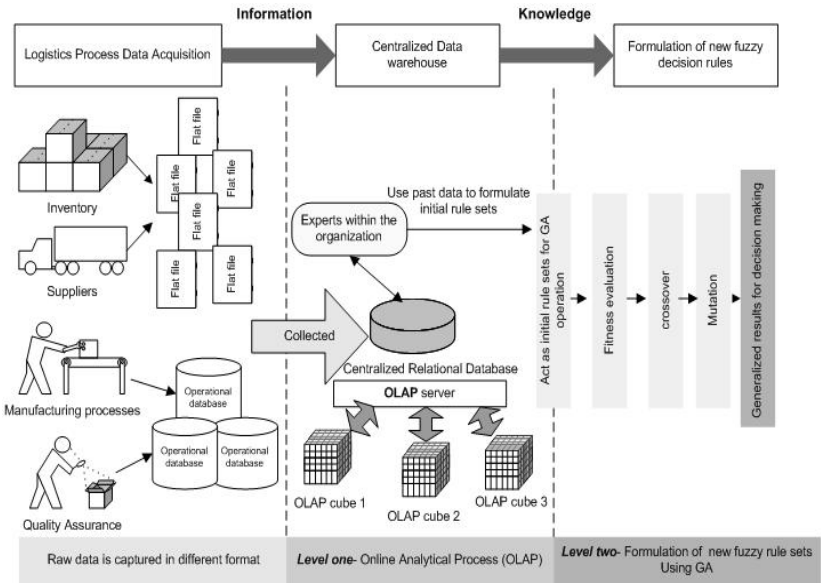


Fig. 1. The proposed framework

The practitioners could freely select the specifically influential performance measures from a large pool of the candidate performance metrics based on the unique condition of the company, leading to the selection of specific postponement strategy amongst all by comparing the weights.

3.2 Nomenclature and Algorithms

Fuzzy concept is used to map the above linguistic decision rules into genes for GA optimisation, and then the decision makers can choose the proper postponement strategies when proposed GA method provides the optimal or nearly optimal solution.

Lemma 1: $C_h = \{1, 2, \dots, M\}$ represents the index set of chromosomes where M is the total number of chromosomes in the population.

Table 1. Notations and definitions of the proposed system

Symbol	Conversion from Gaussian and CGS EMU to SI ^a
F_p	Total number of Corporate Operation indicators (COI)
Q_r	Total number of Customer Satisfaction indicators (CSI)
M_p	Total number of membership functions of COI
M_s	Total number of membership functions of CSI
M	Total number of chromosomes
x	Historical data of an appraisal factor
K	Constant, representing a certain process
T	Ideal target
γ	The specific loss constant
y_j	Parametrical value of the generated rules represented in chromosomes
y_j'	Parametrical value of the test objects
w_j	The weight of the j^{th} parameter
n	Total number of test objects selected
$c_{p_{iu}}$	Center abscissa of the membership function $\tilde{F}_{p_{iu}}$ for COI
$c_{q_{iv}}$	Center abscissa of the membership function $\tilde{F}_{q_{iv}}$ for CSI
$w_{p_{iu}}$	Half the spread of the membership function $\tilde{F}_{p_{iu}}$ for COI
$w_{q_{iv}}$	Half the spread of the membership function $\tilde{F}_{q_{iv}}$ for CSI
L_{p_u}	Linguistics variable for COI
L_{q_v}	Linguistics variable for CSI
P	Index set of FPI, $P=\{1,2,\dots, F_p\}$
Q	Index set of QI, $D=\{1,2,\dots, C_s\}$
p_{iu}	The fuzzy membership function of COI
q_{iv}	The fuzzy membership function of CSI
F_p	Total number of Corporate Operation indicators (COI)

Table 2. Sample appraisal factors

Performance Measurement Indicators (PMI)	
Corporate Operation Indicators (COI)	Cost for customer support Transportation cost Inventory holding cost Labelling/Packaging/Assembling processing cost
Customer Satisfaction Indicators (CSI)	Accuracy of order processing Speed of order handling Provision of after-sale service Responsiveness to urgent deliveries

Table 3. Sample rules

Rule 1	
IF	Inventory holding cost is high AND Labeling processing cost is low AND Packaging processing cost is medium AND Assembling processing cost is medium AND Responsiveness to urgent deliveries is low AND Total lead time is high Rate of Pull postponement is high AND
THEN	Rate of Logistics postponement is high AND Rate of Form postponement is low AND Rate of Price postponement is medium
Rule 2	
IF	Inventory holding cost is medium AND Labeling processing cost is low AND Packaging processing cost is low AND Assembling processing cost is low AND Responsiveness to urgent deliveries is high AND Total lead time is low Rate of Pull postponement is low AND
THEN	Rate of Logistics postponement is high AND Rate of Form postponement is high AND Rate of Price postponement is low

Lemma 2: $G_{m \times t}$ represents a gene matrix generated for the population where

$$G_{m \times w} = \left[\begin{array}{cccc|cccc} p_{11} & p_{12} & \cdots & p_{1a} & q_{11} & q_{12} & \cdots & q_{1b} \\ p_{21} & p_{22} & \cdots & p_{2a} & q_{21} & q_{22} & \cdots & q_{2b} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{m1} & p_{m2} & \cdots & p_{ma} & q_{m1} & q_{m2} & \cdots & q_{mb} \end{array} \right] = ((p_{iu})_{m \times a} (q_{iv})_{m \times b})$$

$$p_{iu} = \text{random}[LL, UL], \forall i \in C_h, \forall u \in P, m = M$$

$$q_{iv} = \text{random}[LL, UL], \forall i \in C_h, \forall v \in Q, m = M$$

$$\text{and } t = a + b, a = F_p, b = Q_r$$

$$p_{i,2u-1} = c_{iu}, \forall i \in C_h, \forall u \in L_{p_u}, m = M \text{ and } e = M_p$$

$$p_{i,2u} = w_{iu}, \forall i \in C_h, \forall u \in w_{p_u}, m = M \text{ and } e = M_p$$

$$q_{i,2v-1} = c_{iv}, \forall i \in C_h, \forall v \in L_{q_v}, m = M \text{ and } n = M_q$$

$$q_{i,2v} = w_{iv}, \forall i \in C_h, \forall v \in w_{q_v}, m = M \text{ and } n = M_q$$

Lemma 3: $K_{1 \times t}$ represents a gene matrix generated for the membership functions associated with the consequence of fuzzy rule $P_{1 \times t}$, respectively, where $6^{P_{iu}}$ represent $1^{F_{P_{iu}}}$ for COI. There are 6 genes in the membership function sets.

$$\tilde{F}_{P_{iu}} = \left\{ c_{P_{iu}} - w_{P_{iu}}, w_{P_{iu}}, c_{P_{iu}}, w_{P_{iu}}, c_{P_{iu}} + w_{P_{iu}}, w_{P_{iu}} \right\}$$

where the linguistic variables are low, medium and high and are equal in width. And

six k_{ij} represents one $\tilde{F}_{q_{iv}}$ for process parameter q_{iv} . There are totally 6 genes in the sets of membership functions.

$\tilde{F}_{q_{iv}} = \{c_{q_{iv}} - w_{q_{iv}}, w_{q_{iv}}, c_{q_{iv}}, w_{q_{iv}}, c_{q_{iv}} + w_{q_{iv}}, w_{q_{iv}}\}$ where the linguistic variables are low, medium and high and are equal in width. These are shown in Fig. 2 with the assumption that the fuzzy rule set be isosceles-triangle. $c_{p_{iu}}$ and $w_{p_{iu}}$ represent the center abscissa and half the spread of the membership function $\tilde{F}_{p_{iu}}$ respectively.

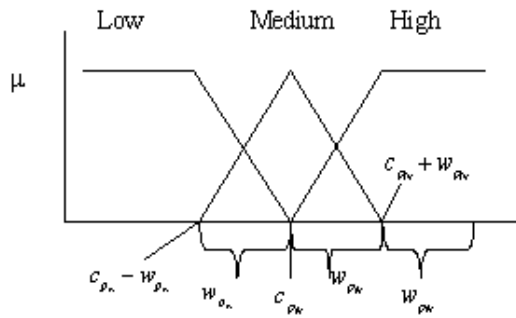


Fig. 2. Membership functions of parameter P_{iu}

The proposed Fuzzy-GA method is used to optimize the fuzzy rule set (i.e. If P_{iu} then q_{iv}) where the associated fuzzy membership function is $\tilde{F}_{p_{iu}}$ and $\tilde{F}_{q_{iv}}$. The genetic algorithm requires a population for a feasible solution to be initialized and that initial population comes from data. In our approach, the fuzzy rule sets are mined from corporation databases, sisters companies, customer specifications or information from the corresponding industrial associations. If the initial number of knowledge sources is not sufficient, some trial data will be encoded and some dummy initial rule sets will be formulated for evolution.

To have a good set of decision variables, the genetic algorithm selects the best chromosome for mating according to the fitness function suggested below. The fitness function is to optimize the accuracy of the fuzzy rules and the performance trade-off function. An evaluation function is used to evaluate the derived fuzzy rule set during the evolution process. Two important factors are used in evaluating derived fuzzy rule sets with their associated membership functions: the accuracy and the performance trade-off of the resulting knowledge structure.

$$\text{Fitness Function} = \frac{\text{Accuracy}}{\text{Performance Trade-off}} \quad (1)$$

$$\text{Accuracy} = \frac{\text{Total number of objects correctly matched by rule set}}{\text{Total number of objects}} \quad (2)$$

Average Performance Trade-off Function (PToF) and Performance Loss Index (PLI) derived from the application of IBLF are proposed for processes associated with the factor-based operations.

$$\text{PToF}(L_i) = 1 + K \left[1 - \left(\frac{x}{T} \right)^\gamma \left(\frac{1-x}{1-T} \right)^\gamma \right] \quad (3)$$

$$\text{Average PLI} = \sum_{i=1}^n (L_i * W_i) \forall i \in n \quad (4)$$

$$\sum_{i=1}^n W_i = W_1 + W_2 + \dots + W_n = 1 \quad (5)$$

Where T = ideal target of the performance indicators;

$K \in (0,1)$, the constant representing various operations

x = the historical data of a influencing factor (e.g. Lead time).

γ = the specific loss constant, which differentiates the loss generated due to the combination of various factors' data.

4 Discussion and Experiment Results

The focus on a parameter-based logistics process, i.e., postponement strategy, arouses interest and further research in this line of study. The new concept of PtoF and PLI are introduced to describe the rationale of postponement strategies. For instance, even though the transportation cost of \$1000 is the lowest, the inventory keeping period may be correspondingly quite long, which may finally cause customer dissatisfaction. As a result, this finance-sensitive strategy is not recommended for obvious reasons. However, if a comparatively higher transportation cost of \$ 1300 is determined, the lead time will be less, and the possibility of customer satisfaction because of getting the goods earlier will be raised. Average PLI could further act as an indicator to show how different combination of the decision variables can affect the corporate profit-making as well as the customer satisfaction.

In order to verify the proposed Fuzzy-GA (FGA) approach, simulations on searching ability were carried out. Two different stochastic-based search methods, Simulated Annealing (SA) and Tabu Search (TS), were used for comparison with the proposed FGA approach. In this experiment, the historical data for supporting the postponement strategy and 50 performance indicators were used for the simulation. The results reported are all averaged over 10 independent runs. In each data set, the best (minimum) fitness value among the 10 simulation runs was documented for the comparison of each search technique mentioned above.

Table 4 presents that ten independent runs of fitness values acquired by various search techniques using 50 performance indicators. According to the experiment, SA was the worst performer in all 10 independent runs and the proposed FGA approach achieved the smallest average object value at 1.09.

Table 4. Best (Minimum) fitness values obtained by FGA, SA AND TS

Number of runs	SA	TS	FGA
1	1.223	1.116	1.013
2	1.137	1.183	1.122
3	1.138	1.101	1.11
4	1.221	1.012	1.098
5	1.201	1.213	1.098
6	1.133	1.128	1.093
7	1.151	1.121	1.101
8	1.162	1.044	1.099
9	1.149	1.159	1.041
10	1.188	1.144	1.123
Average	1.170	1.122	1.09

5 Conclusions

In order to survive in the increasingly customer-oriented marketplace, optimal decision variable settings mark the fastest growing quality organization's success. Due to the inherent complexity of processes, the decision variables are typically chosen by costly, trial-and error prototyping, with the resulting solutions often sub-optimal. Our approach enables decision makers to leverage knowledge in decision variable settings: Fuzzy concept is used for translation of decision rules into genes, while GA with trade-off concern optimises the relevant genes to form the best postponement strategy correspondingly. The research results also show its advantage over SA and TS. The newly-proposed method not only provides a theoretical and scientific way to examine the parameter-based logistics process, but also solves real-life industrial issues. Future work will entail fine-tuning the proposed fitness function, by carefully considering performance trade-off and the accuracy of the fuzzy rules.

Acknowledgement

The authors wish to thank the Research Committee of the Hong Kong Polytechnic University for the support of this research.

References

- [1] Ballou, R.H.: Business Logistics Management: Planning, Organising, and Controlling the Supply Chain, 4th edn. Prentice-Hall International, Englewood Cliffs (1999)
- [2] Lee, H.L.: Design for Supply Chain Management: Concepts and Examples. In: Elwood, S., Buffa, R.K. (eds.) Perspectives in Operations Management: Essays in Honor of Elwood, ch. 3, pp. 45–66. Kluwer Academic Publishers, Boston (1993)
- [3] Whang, S., Lee, H.L.: Value of Postponement. In: Product Variety Management Research Advances, ch. 4, pp. 65–84. Kluwer Academic Publishers, Boston (1999)
- [4] Bitran, G.R., Haas, E.A., Matsuo, H.: Production Planning of Style Goods with High Setup Costs and Forecast Revisions. *Operations Research* 34(2), 226–236 (1986)
- [5] Fisher, M., Raman, A.: Reducing the Cost of Demand Uncertainty Through Accurate Response to Early Sales. *Operations Research* 44(1), 87–99 (1996)

- [6] Alderson, W.: Marketing Efficiency and the Principle of Postponement, Cost and Profit Outlook (3) (1950)
- [7] Bowersox, D.J., Closs, D.J.: Logistical Management: the Integrated Supply Chain Process. Macmillan, New York (1996)
- [8] Lee, H.L., Tang, C.S.: Modeling The Costs And Benefits of Delay Product Differentiation. Management Science 43(1), 40–54 (1997)
- [9] Harvard Business School, Benetton (A) and (B), Harvard Teaching Case 9-685-014, Cambridge, MA (1986)
- [10] Iacocca Institute, 21st Century Manufacturing Enterprise Strategies, Lehigh University, Bethlehem, PA (1991)
- [11] Van Hoek, R.I.: The discovery of postponement: a literature review and directions for research. Journal of Operations Management 19(2), 161–184 (2000)
- [12] Christopher, M.: The agile supply chain: competing in volatile markets. Industrial Marketing Management 29(1), 37–44 (2000)
- [13] Lee, H.L., Whang, S.: Winning the last mile of e-commerce. MIT Sloan Management Review 42(4), 54–62 (2001)
- [14] Yang, B., Burns, N.D., Backhouse, C.J.: Implications of postponement for the supply chain. International Journal of Production Research 41(9), 2075–2090 (2003)
- [15] Agrawal, M.K., Pak, M.H.: Getting smart about supply chain management. The McKinsey Quarterly 2, 22–25 (2001)
- [16] Bowersox, D.J., Closs, D.J.: Logistical Management: the Integrated Supply Chain Process. Macmillan, New York (1996)
- [17] Holland, J.H.: Adaptation in Natural and Artificial Systems. MIT Press, Cambridge (1992)
- [18] Gen, M., Cheng, R.: Genetic algorithms and engineering optimization. Wiley, New York (2000)
- [19] Al-Kuzee, J., Matsuura, T., Goodyear, A., Nolle, L., Hopgood, A.A., Picton, P.D., Braithwaite, N.S.J.: Optimization of plasma etch processes using evolutionary search methods with in situ diagnostics. Plasma Sources Science Technology 13(4), 612–622 (2004)
- [20] Santos, C.A., Spim, J.A., Ierardi, M.C.F., Garcia, A.: The use of artificial intelligence technique for the optimisation of process parameters used in the continuous casting of steel. Applied Mathematical Modelling 26(11), 1077–1092 (2002)
- [21] Li, T.S., Su, C.T., Chiang, T.L.: Applying robust multi-response quality engineering for parameter selection using a novel neural–genetic algorithm. Computers in Industry 50(1), 113–122 (2003)
- [22] Milfelner, M., Kopac, J., Cus, F., Zuperl, U.: Genetic equation for the cutting force in ball-end milling. Journal of Materials Processing Technology 164-165, 1554–1560 (2005)
- [23] Leung, R.W.K., Lau, H.C.W., Kwong, C.K.: An expert system to support the optimization of ion plating process: an OLAP-based fuzzy-cum-GA approach. Expert Systems with Applications 25(3), 313–330 (2003)
- [24] Leung, B.P.K., Spiring, F.A.: The inverted beta loss function: properties and applications. IIE Transactions 34(12), 1101–1109 (2002)
- [25] Taguchi, G.: Introduction to Quality engineering: Designing Quality into Products and processes. Kraus, White Plains, NY (1986)
- [26] Fatikow, S., Rembold, U.: Microsystem Technology and Microrobotics. Springer, Heidelberg (1997)
- [27] van Hoek, R.I.: The rediscovery of postponement a literature review and directions for research. Journal of Operations Management 19(2), 161–184 (2001)
- [28] Yang, B., Burns, N.D., Backhouse, C.J.: The application of postponement in industry. IEEE Transactions on Engineering Management 52(2), 238–248 (2001)

Solving the Delay-Constrained Capacitated Minimum Spanning Tree Problem Using a Dandelion-Encoded Evolutionary Algorithm

Ángel M. Pérez-Bellido¹, Sancho Salcedo-Sanz^{1,*}, Emilio G. Ortiz-García,
Antonio Portilla-Figueras, and Maurizio Naldi²

¹ Department of Signal Theory and Communications, Universidad de Alcalá,
28871 Alcalá de Henares, Madrid, Spain

² Dip. di Informatica, Sistemi e Produzione, Via del Politecnico 1, 00133 Roma, Italy
`sancho.salcedo@uah.es`

Abstract. The Delay-Constrained Capacitated Minimum Spanning Tree (DC-CMST) is a recently proposed problem which arises in the design of the topology of communications networks. The DC-CMST proposes the joint optimization of the network topology in terms of the traffic capacity and its mean time delay. In this paper, an evolutionary algorithm which uses *Dandelion-encoding* is proposed to solve the problem. The Dandelion code has been recently proposed as an effective way of encoding trees in evolutionary algorithms, due to its good properties of locality. We describe the main characteristics of the algorithm, and compare its results with that of an existing heuristic for the DC-CMST. We show that our Dandelion-encoded evolutionary algorithm is able to obtain better results in all the instances tackled.

1 Introduction

The design of the topology is a key point in the design and deployment of computers and communications networks. Topology affects very important points such as communication costs or average time delay of the network [1], [2]. There are several research works about topology design both in backbone and local networks [3,5]. Regarding local networks, these can be modeled as a backbone node (source node) and several trees that cover all end user nodes to satisfy the constraints of traffic volume [1].

In the literature, the works related to topology discovery of local networks can be classified in two main problems: Capacitated Minimum Spanning Tree (CMST) and Delay-Constrained Minimum Spanning Tree (DCMST) problems [6,10]. The

* This work has been partially supported by Universidad de Alcalá and Comunidad de Madrid with a grant number CCG07-UAH/TIC-1894, and by Ministerio de Educación with a grant number TEC 2006-07010/TCM. A. M. Pérez-Bellido and E. G. Ortiz-García are supported by Universidad de Alcalá, under the University F.P.I. grants program.

CMST problem consists of finding a set of minimum spanning trees rooted at the source node which satisfy a set of traffic constraints [8]. This problem is known to be NP-complete [9], and several heuristic approaches have been applied to solve it [6], [7]. On the other hand, the DCMST problem consists of finding the least cost broadcast and multi-cast trees rooted at the source node, which have the minimum total cost among all possible spanning trees, and also have a maximum end-to-end delay bounded by a given delay constraint Δ [10], [11].

In [1] it was shown that the joint optimization of the network topology in terms of the traffic capacity and its mean time delay is needed in order to obtain reasonable quality of service (QoS) rates in the current communication networks. Thus, the Delay-Constrained Capacitated Minimum Spanning Tree (DC-CMST) problem was presented and solved using an ad-hoc heuristic called *Least-cost* (LC) heuristic. The LC heuristic starts from the solution to the CMST problem provided by the Esau and William's algorithm [7], and then applies two different versions of the *Mean Delay Capacity Allocation Algorithm* (MDCAA) in order to obtain feasible solutions for the DC-CMST. In a recent paper [2], the same authors proposed an exact algorithm for the DC-CMST, which provide the optimal solution for the problem. They show that this exact algorithm can compute the DC-CMST solution in reasonable time only for small networks, of less than 30 nodes. Thus, for real-size networks, exact algorithms are computationally very inefficient, and thus heuristics approaches are usually the best option.

In this paper we propose an evolutionary algorithm to solve the DC-CMST which uses a Dandelion encoding to represent the complete local network. The Dandelion encoding has been recently proposed as an effective method to represent trees in evolutionary algorithms [12], with good properties of locality (small changes in the representation make small changes in the tree). This locality makes the evolutionary search more effective than in other tree encodings such as Prüfer encoding [14]. In this paper, we present the different adaptations that must be included in the Dandelion-encoding evolutionary algorithm to solve the DC-CMST. We also compare the results of this approach with those obtained by the LC heuristic in a set of DC-CMST instances of different size. We will show that our evolutionary algorithm for the DC-CMST outperforms the LC heuristic in all the test performed.

The rest of the paper is structured as follows: next section introduces the DC-CMST problem following the description in [1]. Section 3 presents the main characteristics of the Dandelion-encoding evolutionary algorithm proposed in this paper. Special analysis of the encoding, initialization and evolutionary operators are carried out. Section 4 shows the performance of the proposed approach by comparing the results obtained in different DC-CMST instances with that of an existing heuristic. Section 5 closes the paper giving some final remarks.

2 Problem Formulation

The DC-CMST problem has been recently proposed in [1]. In the formulation of the DC-CMST problem, a set of assumptions must be made:

1. There is only one source node, which has unlimited capacity.
2. The traffic generated at a given node (q_i) cannot exceed the maximum traffic covered by one tree ($\max(q_i) \leq MAX, i = 1, \dots, n$).
3. The traffic at a given node is not splintered.
4. The total traffic exceeds the maximum traffic served by one of the trees conforming the network ($\sum_{i=1}^n q_i > MAX$).
5. The arrival of packets follows a Poisson process.
6. The service time of packets is exponentially distributed.

With these assumptions, our formulation of the DC-CMSTP follows the one given in [1]: Obtaining a tree network with minimum total link cost, which satisfies that the mean network delay time (T) is bounded by a given time (Δ), and also that the traffic capacity limitation is less than a given value MAX in each subtree. Figure 1 shows an example of the problem definition. In order to provide a mathematical formulation of the DC-CMST problem, we need first to define several parameters of the problem. Let T be the mean time delay of the network, it can be defined as:

$$T = \frac{1}{\nu} \sum_{k=1}^n \lambda_k T_k \quad (1)$$

where ν stands for the total traffic of the network (traffic between all source-destination pairs), λ_k is the traffic flow on link k in the current topology (packets/s) and T_k is the mean delay time of link k in the current topology. Since each link k can be regarded as an independent $M/M/1$ queue, the mean delay T_k of link k is given by the following equation [1]:

$$T_k = \frac{1}{(\mu \cdot C_k - \lambda_k)} \quad (2)$$

where $\frac{1}{\mu}$ is the average packet length (bits/packet), and C_k stands for the capacity of link k in the current topology (bits/s). If we include this expression in (1), we obtain the following equation for the total network mean time delay T :

$$T = \frac{1}{\nu} \sum_{k=1}^n \lambda_k \left(\frac{1}{(\mu \cdot C_k - \lambda_k)} \right) \quad (3)$$

The DC-CMST problem can now be mathematically stated, in the following way:

$$\min \left(D_{cost} = \sum_{k=1}^n d \cdot C_k \cdot d_k \right) \quad (4)$$

subject to:

$$\frac{\lambda_k}{\mu} \leq C_k \quad (5)$$

$$T = \frac{1}{\nu} \sum_{k=1}^n \lambda_k \left(\frac{1}{(\mu \cdot C_k - \lambda_k)} \right) \leq \Delta \quad (6)$$

$$\sum_{i \in R_j} q_i \cdot x_{ij} \leq MAX \quad (7)$$

$$\sum_{i,j} x_{ij} = n \quad (8)$$

$$x_{ij} \in \{0, 1\} \quad (9)$$

where, the objective function of the DC-CMSTP is given by Equation (4), and consists of finding a collection of trees with minimal link cost (D_{cost}). Note that D_{cost} depend on the values of C_k (capacity of link k in the current topology (in seconds), d unit cost of link capacity and d_k defined as the distance between node k in a given tree and its antecessor node in that tree (note that this distance may be different depending on the considered tree). The constraints of the DC-CMSTP are the following: the average traffic flow on a link must be smaller than the capacity of the link, Equation (5). The mean delay of the network has to be dropped within allowable mean delay time (Δ), Equation (6). The total traffic flow in one tree (R_j) must be below a value MAX , Equation (7). Finally, Equation (8) ensures that n nodes are included in the final solution. Note that variable x_{ij} is 1 if there is a link between nodes i and j in the current topology, and 0 otherwise, as Equation (9) states.

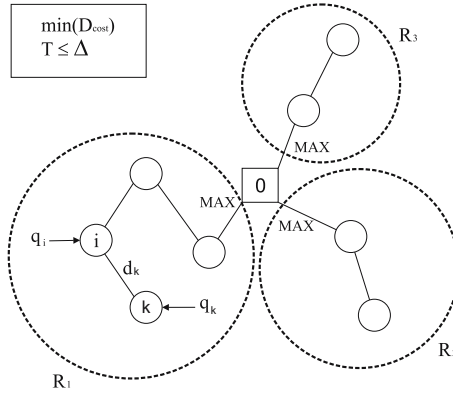


Fig. 1. An example of the DC-CMST problem

3 A Dandelion-Encoded Evolutionary Algorithm for the DC-CMSTP

3.1 Algorithm Encoding

The Dandelion code is a *Cayley-like* encoding [12] which has been recently described and used for encoding trees in genetic algorithms [12], [13]. There are

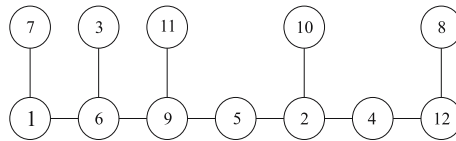
several decoding algorithms (string to tree) for the Dandelion code. In this paper we use the so-called *fast algorithm*, proposed by Picciotto in [15], which has been also used in [13]:

- **Input:** A Dandelion code $C = (c_2, c_3, \dots, c_{n-1})$.
- **Output:** The tree $T \in \mathcal{T}_n$ corresponding to C .
- **Step 1:** Define the function $\phi_C : [2, n-1] \rightarrow [1, n]$ such that $\phi_C(i) = c_i$ for each $i \in [2, n-1]$.
- **Step 2:** Calculate the cycles associated to the function ϕ_C , $Z_1, Z_2, \dots, Z_k$. Let b_i be the maximum element in cycle Z_i . We assume that the cycles are recorded such that b_i is the rightmost element of Z_i , and that $b_i > b_j$ if $i < j$.
- **Step 3:** Form a single list π of the elements in $Z_1, Z_2, \dots, Z_k$, in the order they occur in this cycle list, from the first element of Z_1 to the last element of Z_k .
- **Step 4:** Construct the tree $T \in \mathcal{T}_n$ corresponding to C in the following way: take a set of n isolated vertices (labeled with the integers from 1 to n), create a path from vertex 1 to vertex n by following the list π from left to right, and then create the edge (i, c_i) for every $i \in [2, n-1]$ which does not occur in the list π .

We will illustrate this fast algorithm using the example in Figure 2. This figure proposes the Dandelion code $C = (4, 6, 2, 5, 9, 1, 12, 6, 2, 9)$. Note that there are three cycles in this case, $Z_1 = (6, 9)$, $Z_2 = (5)$ and $Z_3 = (2, 4)$. Note also the order in which we have recorded these cycles follows the indications in the step 2 of the fast decoder algorithm. We form then the list $\pi = [6, 9, 5, 2, 4]$, and construct the first part of the tree T starting from vertex 1, ending in vertex 12, and following the numbers in π . The rest of the tree is constructed by creating the corresponding edges (i, c_i) for i which are not in the list π , in this case the vertices 7, 3, 11, 10 and 8.

(4, 6, 2, 5, 9, 1, 12, 6, 2, 9)

(a)



(b)

Fig. 2. (a) Example of a Dandelion code; (b) The final tree after the decoding process

3.2 Initialization

In order to initialize the algorithm, we start from the minimum spanning tree (MST) generated by the initial graph formed by all the nodes of the network (fully connected). We use the Prim's algorithm [17] to obtain the MST. Note that in this case we only consider the distances between nodes, but no traffic figures. The next step is to convert the MST to its corresponding Dandelion code, following the tree-to-code procedure described in [12]:

- **Input:** The MST (from Prim's algorithm) of the DC-CMST problem.
- **Output:** A Dandelion code $C_{MST} = (c_2, c_3, \dots, c_{n-1})$ corresponding to the MST.
- **Step 1:** Find the unique path from 1 to n in T , and let π be the ordered list of intermediate vertices.
- **Step 2:** Recover the cycles $\{Z_i\}$ by writing π in a left-to-right list, and closing the cycle immediately to the right of each right-to-left minimum.
- **Step 3:** The Dandelion code corresponding with T is the unique code $C_{MST} = (c_2, c_3, \dots, c_{n-1})$ such that: a) the cycles of the function $\phi_C(i) = c_i$ are $\{Z_i\}$, where $\phi_C(i)$ stands for the value of the i th position in string C_{MST} ; b) for each $i \in [2, n-1]$ which does not occur in π the first vertex on the path from vertex i to vertex n in the tree T is c_i .

Using this code C_{MST} , we generate the initial population of our Dandelion-based evolutionary algorithm by applying mutations to C_{MST} (we call C'_{MST} to a given mutation of C_{MST}), in such a way that $C'_{MST}(i) \in [2, n-1]$.

3.3 Evolutionary Operators

Our evolutionary algorithm is structured in the traditional form of a classic genetic algorithm [16], with procedures of Selection, Crossover and Mutation. The Selection procedure is the standard roulette wheel, in which the probability of survival of a given individual is proportional to its fitness value. A two-points Crossover operator is applied, where the parents are randomly chosen among the individuals in the population. The Mutation operator changes specific values of C , substituting its value by a different one in $[2, n-1]$. Regarding the parameters of the algorithm, we have implemented the standard values of Crossover ($P_c = 0.6$) and Mutation ($P_m = 0.01$) operators. The population size is 100 and the generations number has been fixed to 1000, after which we keep the best tree encountered so far.

In addition to the standard operators, we include a new operator consisting in applying the Prim's algorithm over each sub-tree of the individuals, in every generation of the algorithm. Since the objective function consists of a sum involving the distances between nodes belonging to the same sub-tree, it is expected that this procedure improves the quality of the final solution of the algorithm.

3.4 Constraints Requirements of the DC-CMST and Objective Function

In [1] is shown that if we consider a restriction to the DC-CMST as:

$$\min \left(D_{cost} = \sum_{k=1}^n d \cdot C_k \cdot d_k \right) \quad (10)$$

subject to:

$$\frac{1}{\nu} \sum_{k=1}^n \left(\frac{\lambda_k}{\mu \cdot C_k - \lambda_k} \right) = \Delta \quad (11)$$

values of C_k can be calculated as

$$C_k = \frac{\lambda_k}{\mu} \left(1 + \frac{\sum_{j=1}^n \sqrt{d \cdot \lambda_j \cdot d_j}}{\nu \cdot \Delta \cdot \sqrt{d \cdot \lambda_k \cdot d_k}} \right), \quad (12)$$

note that if we use these values of C_k , we ensure that the generated tree always fulfils Constraints (5) and (6), and we only have to manage Constraint (7) in our evolutionary algorithm.

The objective function of our Dandelion evolutionary algorithm is given next. First we define the following parameter:

$$\mathcal{E} = \prod_j \left(\frac{1}{MAX} \sum_{i \in R_j} q_i \cdot x_{ij} \right), \quad (13)$$

where j is the set of values such that $\sum_{i \in R_j} q_i \cdot x_{ij} > MAX$, and then the objective function used in this paper for the DC-CMST is:

$$f(T) = \sum_{k=1}^n d \cdot C_k \cdot d_k \cdot \mathcal{E} \quad (14)$$

4 Experiments and Results

In order to test the performance of our proposal, we have carried out several experiments similar to the one shown in [1]. A set of DC-CMST instances have been constructed, strictly following the indications in [1]. Specifically, we have considered 15 instances of the DC-CMST problem, in networks of 30, 50 and 70 nodes, 5 networks in each case (see [1] for information about the computational instances tackled). We compare the result obtained by our Dandelion-encoded evolutionary algorithm with that of the LC heuristic presented in [1]. Table 1 reports the details of this comparison. Note that the LC algorithm is deterministic, so one value for each instance is provided. On the other hand, our EA algorithm

Table 1. Results of the Dandelion-encoded EA proposed, and comparison with the results obtained by the LC algorithm in [1]

Instance #	Network size (nodes)	LC algorithm in [1]	Dandelion EA		
			Best	Mean	Std. Dev.
1	30	2158.86	2139.52	2152.58	12.65
2	30	1926.01	1751.49	1772.10	20.66
3	30	1855.45	1671.56	1721.30	32.31
4	30	2147.29	2086.38	2120.09	20.87
5	30	1844.00	1787.00	1787.24	0.76
6	50	3136.46	3015.61	3085.15	40.73
7	50	3550.55	3449.32	3503.89	22.47
8	50	2956.75	2709.80	2797.22	63.07
9	50	2488.86	2414.22	2452.38	16.11
10	50	2479.62	2299.15	2356.25	51.82
11	70	5403.27	5293.44	5320.54	14.77
12	70	3899.74	3693.69	3736.63	24.94
13	70	4960.57	4800.59	4847.00	33.23
14	70	3842.11	3639.38	3695.26	55.49
15	70	5826.47	5707.86	5835.26	68.71

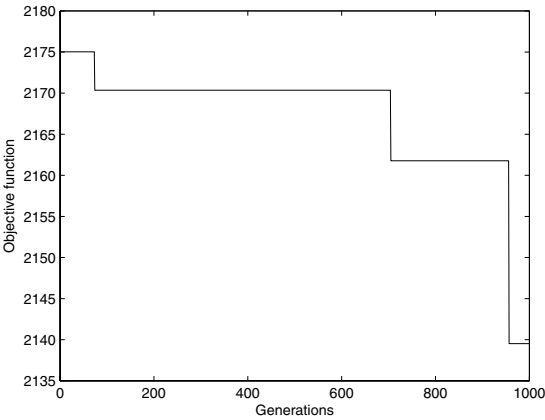


Fig. 3. Evolution of the EA in DC-CMST instance #1

has been launched 30 times, and we provide the best, mean and standard deviation values for each instance. Our EA has been run using a population of 100 individuals during 1000 generations.

The results shown in Table 1 confirm the good performance of our approach. The Dandelion-encoded EA is able to obtain better results than the LC algorithm in all cases considered. In all the instances tackled, the best value found by our Dandelion-encoded EA is better than the result obtained by the LC algorithm. Moreover, we find that in all instances considered but one (instance 15), the

mean value of the 30 runs with the EA is better than the result obtained by the LC algorithm. The differences between the proposed EA and the LC algorithm are larger as the instances' size grows. This means that our EA is more scalable than the LC algorithm.

Figure 3 shows the evolution of the best run for DC-CMST instance #1. Note that the algorithm's evolution has the form of steps, with flat zones between improvements. This behavior is consistent with the algorithm proposed: recall that we include a MST local search in each sub-branch of the tree, so small improvements in the objective function (associated with small changes in sub-branches) are discarded. The improvements are obtained when individuals with a better structure appear, which will be quite different from existing individuals.

5 Conclusions

In this paper we have presented a Dandelion-encoding evolutionary algorithm for the Delay-Constrained Capacitated Minimum Spanning Tree (DC-CMST) problem. This problem arises in the topological design of communication networks, and is complicated because it considers optimization of the network topology in terms of the traffic capacity and its mean time delay. Due to this complexity, only heuristic approaches have been applied for large instances of this problem. Specifically, the best algorithm to solve the DC-CMST is known as the Least-Cost heuristic (LC). In the paper we show that the Dandelion-encoding algorithm we propose is able to improve the results of the LC heuristic in several DC-CMST problem instances.

References

1. Lee, Y.J., Atiquzzaman, M.: Least cost heuristic for the delay-constrained capacitated minimum spanning tree problem. *Computer Communications* 28, 1371–1379 (2005)
2. Lee, Y.J., Atiquzzaman, M.: Exact algorithm for delay-constrained capacitated minimum spanning tree network. *IET Communications* 1(6), 1238–1247 (2007)
3. Thomadsen, T., Larsen, J.: A hub location problem with fully interconnected backbone and access networks. *Computers & Operations Research* 34, 2520–2531 (2007)
4. Astic, I., Festor, O.: A hierarchical topology discovery service for IPv6 networks. In: *Proc. of the IEEE/IFIP Network Operations and Management Symposium*, pp. 497–510 (2002)
5. Bejerano, Y., Breitbart, M., Rastogi, R.: Physical topology discovery for large multi subnet networks. In: *Proc. of IEEE INFOCOM 2003*, pp. 342–352 (2003)
6. Gavish, B.: Parallel savings heuristic for the topological design of local access tree networks. In: *Proc. of IEEE INFOCOM 2003*, pp. 130–139 (2003)
7. Esau, L., Williams, K.: On teleprocessing system design, part II. *IBM System Journal* 3, 142–147 (1996)
8. Chandy, K.M., Lo, T.: The capacitated minimum spanning tree. *Networks* 3, 173–181 (1973)
9. Papadimitriou, C.: The complexity of the capacitated minimum spanning tree problem. *Networks* 8, 217–230 (1978)

10. Karaman, A., Hassanein, H.: DCMC—delay constrained multipoint communication with multiple sources. In: Proc. of the IEEE International Symposium on Computers and Communications (2003)
11. Reeves, D.S., Salama, H.F.: A distributed algorithm for delay-constrained unicast routing. *IEEE/ACM Transactions on Networking* 8(2), 239–250 (2000)
12. Paulden, T., Smith, D.K.: From the Dandelion code to the Rainbow code: a class of bijective spanning tree representations with linear complexity and bounded locality. *IEEE Trans. Evol. Comput.* 10(2), 108–123 (2006)
13. Thompson, E., Paulden, T., Smith, D.K.: The Dandelion code: a new coding of spanning trees for genetic algorithms. *IEEE Trans. Evol. Comput.* 11(1), 91–100 (2007)
14. Deo, N., Micikevicius, P.: Prüfer-like codes for labeled trees. *Congressus Numerantium* 151, 65–73 (2001)
15. Picciotto, S.: How to encode a tree, Ph.D. dissertation, Univ. California, San Diego (1999)
16. Goldberg, D.: Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading (1989)
17. Prim, R.C.: Shortest connection networks and some generalisations. *Bell System Technical Journal* 36, 1389–1401 (1957)

Generalized Extremal Optimization for Solving Multiprocessor Task Scheduling Problem

Piotr Switalski¹ and Franciszek Seredynski^{2,3}

¹ The University of Podlasie, Computer Science Department, 3 Maja 54,
08-110 Siedlce, Poland

² Polish-Japanese Institute of Information Technology, Koszykowa 86,
02-008 Warsaw, Poland

³ Institute of Computer Science, Polish Academy of Sciences, Ordona 21,
01-237 Warsaw, Poland

Abstract. In this paper we propose a solution of a multiprocessor task scheduling problem with use of a new meta-heuristic inspired by a model of natural evolution called Generalized Extremal Optimization (GEO). It is inspired by a simple co-evolutionary model based on a Bak-Sneppen model. One of advantages of the model is a simple implementation of potential optimization problems and only one free parameter to adjust. The idea of GEO metaheuristic and the way of applying it to the multiprocessor scheduling problem are presented in the paper. In this problem the tasks of a program graph are allocated into multiprocessor system graph where the program completion time is minimized. The problem is known to be a NP-complete problem. In this paper we show that GEO is able to solve this problem with better performance than genetic algorithm.

Keywords: multiprocessor task scheduling problem, Generalized Extremal Optimization, GEO, genetic algorithm.

1 Introduction

In the present-day many optimization problems in science and engineering [1] are difficult to solve. These problems are often NP-complete problems [2]. The problems belong to the class of computational problems for which no efficient solution algorithm has been found. NP-complete problems have been only solved approximately by existing techniques like randomization, parameterization or using heuristics (meta-heuristics). The most methods based on local search algorithms in a complex problem with multiple local optima often converge to the local minimum [3].

A more general approach is to use a global search algorithm. In this case we can find global optimum, but it requires more computational cost e.g. time for solving optimization problem. One of class of the global optimization algorithms is particularly worth to consider - algorithms based on natural phenomena. This motivation is based on observation of natural processes which are frequently self-optimized. The most commonly used algorithms are Genetic Algorithm (GA)

[4], Simulated Annealing algorithm [5], Swarm-based Optimization algorithm [6], Artificial Immune System algorithm [7].

The multiprocessor task scheduling problem is one of NP-complete problems. The objective of scheduling is to minimize the completion time of parallel application divided into tasks by properly allocating the tasks to the processors [8]. In this problem the tasks organized in a graph need to be allocated into a multiprocessor system graph. The optimal solution of this problem is very difficult to obtain. This problem has been often solved by algorithms based on a natural phenomena mentioned above.

In this paper we have proposed a relatively new metaheuristic called GEO to solve the multiprocessor scheduling problem. Experiments show that this algorithm is very efficient for this problem and provides better results than GA.

The paper is organized as follows. In the next section we describe the multiprocessor task scheduling problem. Section 3 presents GEO algorithm applied for the scheduling problem. Next, in Section 4 experimental results are given. Last section concludes the paper.

2 Multiprocessor Task Scheduling Problem

The problem is defined as follows (see, e.g.[9]). For a given number of processors m and a given topology of connections between them, and a parallel program graph consisting of n tasks with precedence relations and the run times on individual processors, we have to find schedule which has the shortest run time. The schedule must define for each task a time of execution and the processor to be used for execution of the particular task. A topology of multiprocessor system is represented by weighted and directed graph $G_s = (V_s, E_s)$ with m vertices $V_s = \{v_0, \dots, v_{m-1}\}$ and a number of edges $e_{ij} \in E_s$. The vertices represent processors $P_s = \{P_0, \dots, P_{m-1}\}$. Edges are connections between nodes representing processors. Edge e_{ij} is the direct connection between nodes representing processor P_i and processor P_j . All connections (channels of data exchange) are bi-directional. Graph G_s is called a system graph. Fig. 1a presents a two-processors system in architecture called FULL2.

Let's consider program Z consisting of n indivisible tasks $Z = (z_0, \dots, z_{n-1})$. If the results of realization of problem z_i are input data of task z_j (task z_i must be completed before task z_j is started), then these tasks are in precedence relations: $z_i \prec z_j$. If program Z runs on a multiprocessor system, then tasks which are not in a precedence relation can be run simultaneously on different processors. With every task z_i is related processing time b_i (computational cost). Additionally, transfer of task z_i results to task z_j may be related with noticeable data transfer time through the communication channel between processors, if tasks are run on different processors. For example, computational cost may be proportional to the size of these results. Therefore, for every precedence relation between tasks z_i and z_j in program Z communication cost of sending of results from z_i to z_j is defined if they are run on neighboring processors. In other cases, the cost is proportional to the shorter distance $hops_{ij}$ between the processors i and j and

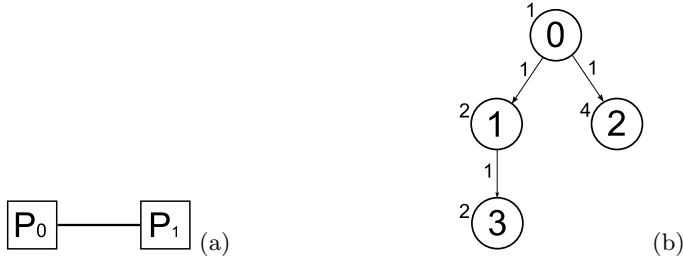


Fig. 1. Program and system graphs: the graph of two-processor system in FULL2 architecture (a), an example of a program graph (b)

it is equal to $a_{ij} * hops_{ij}$, where $hops_{ij}$ is a number of direct links to cross from i to j , a_{ij} is a cost of sending results from one node to another. In particular, the cost equals zero, when the tasks are executed on the same processor. Program Z is represented by weighed, directed and acyclic graph $G_p = (V_p, E_p)$ whose vertices v_i are tasks z_i and edges e_{kl} reflect the precedence relations. Graph G_p is called a program graph or precedence task graph. Fig. 1b presents a precedence graph for a four tasks in a precedence relation.

It is assumed that all processors have the same computational power and communication via links does not consume any processor time. The purpose of scheduling is to distribute task among processors (Fig. 1a) in such a way to minimize the total execution of time T .

3 The Generalized Extremal Optimization Algorithm

3.1 Idea of the Algorithm

The idea of this algorithm is based on Bak-Sneppen model [10]. Evolution in this model is driven by a process in which the weakest species in the population, together with its nearest neighbors, is always forced to mutate. The dynamics of this extremal process show characteristics of Self-Organized Criticality (SOC), such as punctuated equilibrium, that are also observed in natural ecosystems. Punctuated equilibrium is a theory in evolutionary biology. It states that in evolution there are periods of stability punctuated by a change in environment that forces relatively rapid adaptation by generating "avalanches". The probability distribution of these avalanches is described by a power law in the form $p_i = k_i^{-\tau}$, where p is a probability of mutation of i -th individual, k is a position of individual in a rank, τ is a positive parameter. If $\tau \rightarrow 0$ the algorithm search a solution randomly, while $\tau \rightarrow \infty$ the algorithm provides deterministic searching. This idea is used in this algorithm. Bak and Sneppen developed a simplified model of an ecosystem in which N species are placed side by side on a line. Fig. 2 shows the population of species in the Bak-Sneppen model and the idea of GEO algorithm. In the GEO approach, a population of species is a string of bits that encodes the design variables of the optimization problem, and

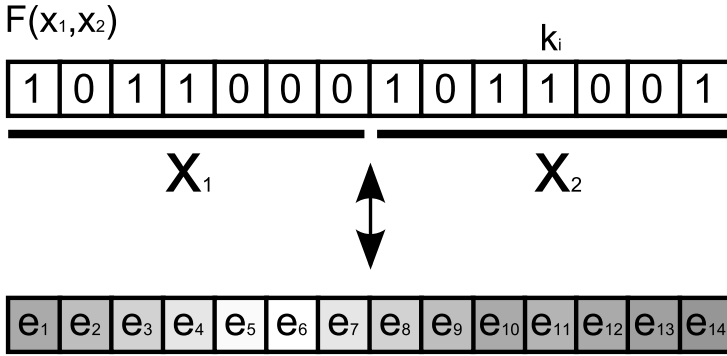


Fig. 2. Population of the species in the Bak-Sneppen model and its correspondence in the GEO algorithm

each bit corresponds to one species. In Fig. 2 two variable function $F(x_1, x_2)$ is optimized. Each variable is coded using seven bits, so the population of the algorithm consisting of 14 bits (upper part of Fig. 2). Each bit of the string is considered as the species (lower part of Fig. 2) of the Bak-Sneppen model. The each bit representing species e_i ($i = 1, N$) has a value 0 or 1. A number of bits per variable depends on the type of the problem. In contrast to GA, in GEO there is not a population of strings, but one population of bits represented by one string. In this algorithm each bit is forced to mutate with a probability proportional to its fitness. The fitness is a number assigned to each bit of this string that indicates the level of adaptability of each bit of the population, according to the gain or loss to value of the fitness function if the bit is mutated.

3.2 Presentation of the Algorithm

According to the paper by [11] the GEO algorithm can be described as follows:

1. Initialize randomly a binary string of length L that encodes N design variables of bit length L/N .
2. For the current configuration C of bits, calculate the objective function value V and set $C_{best} = C$ and $V_{best} = V$.
3. For each bit i do,
 - (a) flip the bit (from 0 to 1 or 1 to 0) and calculate the objective function value V_i of the string configuration C_i ,
 - (b) set the bit fitness F_i as $(V_i - R)$, where R is a constant. It serves only as a reference number and can assume any value. The bit fitness indicates the relative gain (or loss) that one has in mutating the bit.
 - (c) return the bit to its original value.
4. Rank the N bits according to their fitness values, from $k = 1$ for the least adapted bit to $k = N$ for the best adapted. In a minimization problem higher values of F_i will have higher ranking. Otherwise for maximization problems.

If two or more bits have the same fitness, rank them in random order, but following the general ranking rule.

5. Choose a bit i to mutate according to the probability distribution $P_{ik} = k^{-\tau}$, where τ is an adjustable parameter.
6. Set $C = C_i$ and $V = V_i$.
7. If $F_i < F_{best}$ ($F_i > F_{best}$, for a maximization problem) then set $F_{best} = F_i$ and $C_{best} = C_i$.
8. Repeat steps 3 to 7 until a given stopping criteria is reached.
9. Return C_{best} and F_{best} .

4 Experiments

4.1 Representation of the Individual

We implemented the GEO algorithm to a multiprocessor task scheduling problem. The population consists of one binary string. The number of bits in the string is equal to:

$$N * \log_2 M,$$

where N - a number of tasks in a program graph and M - a number of processors. For example, for eight processors (from 0 to 7) we need three bits to represent each processor, so $\log_2 8 = 3$ and for a program graph consisting of $N = 8$ tasks the total number of binary string is equal to 24.

4.2 Adjustment of a τ Parameter

We implemented GEO with the representation of a population described above. The τ parameter influences substantially on the results. In the Fig. 3 we can notice influence of this parameter for obtained results. For small program graphs ($N \leq 20$) (see, Fig. 3a) the best value of τ parameter is 1.0. For larger program graphs (see, Fig. 3b) the τ parameter should be increased from 1.5 to 2.5. Above of these values the algorithm gives only worse results. For the most carried out experiments the τ parameter in the range from 1.5 to 2.5 was an optimal value.

4.3 Two-Processor Scheduling with GEO

A number of experiments with deterministic program graphs known in the literature (see, e.g., [9]) and random program graphs [12] have been conducted. The results were compared with those obtained with use of GA. We used the following parameters in the experiment: τ parameter = 1.5 - 2.0 (GEO algorithm), population size = 200, mutation probability = 0.001, crossover probability = 0.9 (GA algorithm). We assumed an equal number of evaluations of fitness function for both algorithms.

The first program graph used in experiments is *tree15*. It is a binary tree consisting of 15 tasks. All computational and communication costs are the same and equal to 1. The optimal response time T for *tree15* in the two-processor

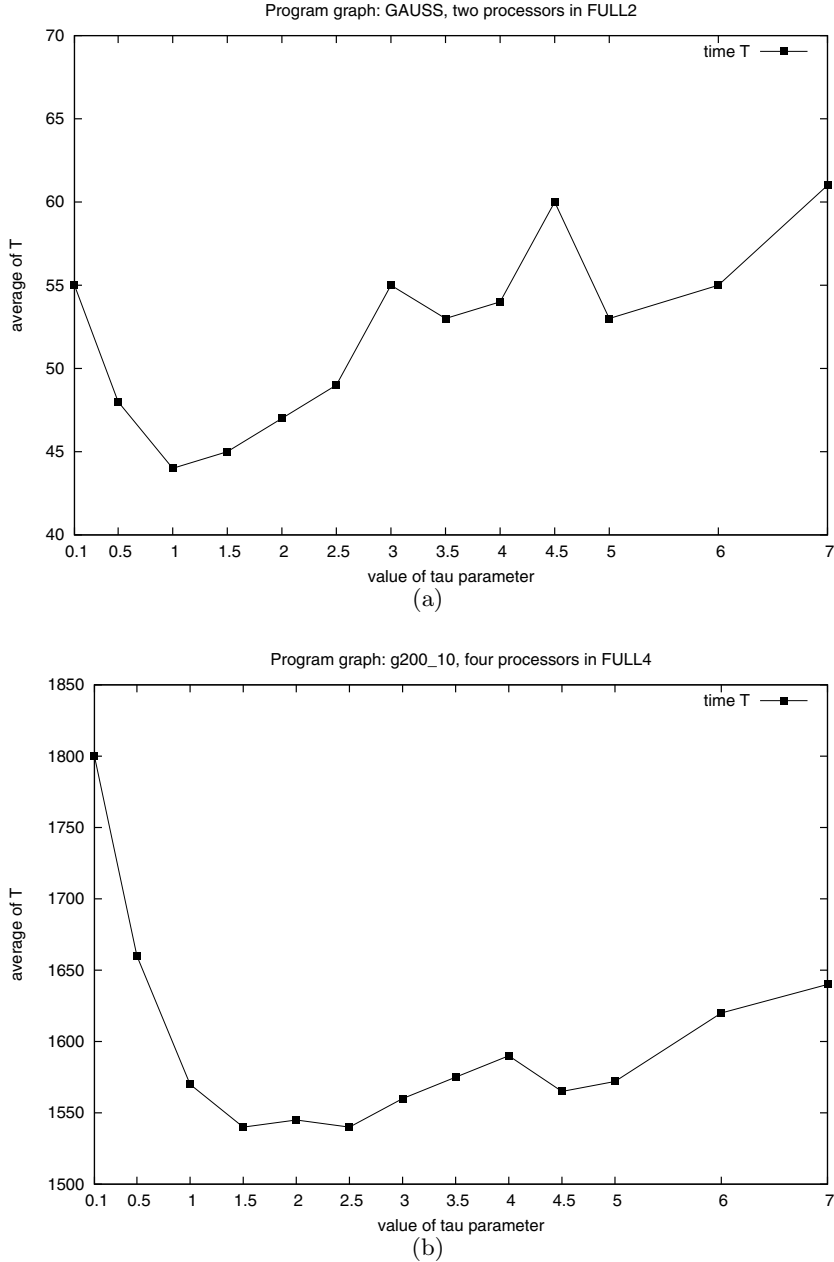


Fig. 3. Influence of the τ parameter on the results for (a) a small program graph, (b) a large program graph (averaged on 10 runs)

Table 1. Comparison of the algorithms: GEO and GA for several program graphs. The best and average (in rounded brackets) values for 10 runs. The optimal solutions are in bold.

Program graph	A number of tasks	GEO	GA
<i>tree15</i>	15	9 (9)	9 (9)
<i>g18</i>	18	46 (46)	46 (46)
<i>g40</i>	40	80 (80)	80 (80)
<i>gauss18</i>	18	44 (44)	44 (46)
<i>g25_1</i>	32	495 (495)	495 (495)
<i>g25_5</i>	18	98 (98)	99(99)
<i>g25_10</i>	21	62 (62)	62 (70)
<i>g100_1</i>	100	1481 (1481)	1481 (1481)
<i>g100_5</i>	120	395(398)	419(404)
<i>g100_10</i>	74	172(177)	178(182)
<i>g200_1</i>	207	3025 (3025)	3025 (3025)
<i>g200_5</i>	191	556(559)	566(569)
<i>g200_10</i>	250	483(490)	507(516)
<i>g400_1</i>	439	6265 (6265)	6265 (6265)
<i>g400_5</i>	406	1228(1229)	1240(1251)
<i>g400_10</i>	288	541(552)	573(594)

system equals 9. Experiments (Tab. 1) have shown that for this program graph all algorithms have found an optimal solutions. The next graphs (*g18* and *g40*) are also simple example of program graphs. The gauss program graph is more difficult because of its nonregular structure. Not all algorithms found optimal solution for this graph. GEO found optimal solution in every run in opposite to GA, where optimal solution was found in only one run for 10 runs.

In the next experiments we used a random program graphs. In the experiments we changed τ parameter to 2.0 for GEO algorithm. Experiments with a relative small graph (*g25*) shows that only GA not found solution for *g25_5*. In the larger random graphs (*g100*, *g200* and *g400*) both algorithms found optimal solutions only for simple variant of these graphs: *g100_1*, *g200_1* and *g400_1*, where an average communication cost equal to an average computational cost. In the other variants, experiments shows that GEO is considerably better than GA.

4.4 Multiprocessor Scheduling with GEO

In this section we present results of GEO algorithm for more than two processors. This variant is more complicated, because the individual in GEO is two times (for four processors) or three times (for eight processors) longer than for two processors.

We analyzed difficult examples of program graphs. Firstly a gauss graph has been tested (see, Tab. 2). Although this graph is represented only by 18 tasks, it is difficult to solve. GEO found optimal results for all cases - a four and eight processors. GA found this result only in a simpler case - for four processors.

Table 2. Comparison of the algorithms: GEO and GA for several program graphs and $k=4$ processors and $k=8$ processors. The best and average (in rounded brackets) values for 10 runs. The optimal solutions are in bold.

Program graph	A number of tasks	k=4		k=8	
		GEO	GA	GEO	GA
<i>gauss18</i>	18	44 (46)	44 (46)	44 (49)	44 (53)
<i>g25_10</i>	21	62 (67)	70 (77)	71 (76)	79 (91)
<i>g100_10</i>	74	176(188)	194(201)	202(205)	206(208)
<i>g200_10</i>	250	485(487)	498(501)	482(489)	483(484)
<i>g400_10</i>	288	495(500)	495(498)	520(525)	521(530)

Afterward we used random graphs (*g25_10*, *g100_10*, *g200_10* and *g400_10*), where an average communication cost is higher than an average computational cost. There are the most difficult cases of these graphs. We compared results of GEO and GA for this graphs (see, Tab. 2). We can notice that GEO is generally better than GA, especially for *g25_10*, *g100_10* and *g200_10*. For the largest program graph *g400_10* GEO found the same or better results.

5 Conclusions

Applying the GEO algorithm to the task scheduling problem has confirmed that this algorithm is useful for this problem. The results of the experiments show advantages of GEO. Simplicity of implementation of the algorithm is one of the advantages. In the opposite to GA we have only one parameter to adjust. Results of the experiments for GEO algorithm are better than GA for optimal parameters of both algorithms.

References

- [1] Pardalos, P.M., Romeijn, H.E.: Handbook of Global Optimization, vol. 2. Springer, Heidelberg (2002)
- [2] Garey, M.P., Johnson, D.S.: Computers and intractability - a guide to NP-completeness. W.H. Freeman and Company, San Francisco (1979)
- [3] Eldred, M.S.: Optimization Strategies for Complex Engineering Applications. Sandia Technical Report SAND98-0340 (1998)
- [4] Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading (1989)
- [5] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by Simulated Annealing, vol. 220(4598), pp. 671–680. ACM, New York (1983)
- [6] Kennedy, J., Eberhart, R.C.: Particle Swarm Optimization. In: Proc. IEEE International Conference on Neural Networks, pp. 1942–1948 (1995)
- [7] Dasgupta, D.: Artificial Immune Systems and Their Applications. Springer, Berlin (1999)

- [8] Kwok, Y., Ahmad, I.: Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors. *ACM Computing Surveys* 31(4), 406–471 (1999)
- [9] Seredynski, F., Zomaya, A.Y.: Sequential and Parallel Cellular Automata-Based Scheduling Algorithms. *IEEE Trans. Parallel Distrib. Syst.* 13(10), 1009–1023 (2002)
- [10] Bak, P., Sneppen, K.: Punctuated equilibrium and criticality in a simple model of evolution. *Phys. Rev. Lett.* 71, 4083–4086 (1993)
- [11] Sousa, F.L., Ramos, F.M., Galski, R.L., Muraoka, I.: Generalized Extremal Optimization: A New Meta-heuristic Inspired by a Model of Natural Evolution. *Recent Developments in Biologically Inspired Computing*, 41–60 (2004)
- [12] Bollobas, B.: *Random Graphs*. Academic Press, New York (1985)

Improving NSGA-II Algorithm Based on Minimum Spanning Tree

Miqing Li, Jinhua Zheng, and Jun Wu

Institute of Information Engineering, Xiangtan University,
411105, Hunan, China

limit1008@126.com, jhzheng@xtu.edu.cn, Jun_wu_521@163.com

Abstract. Diversity maintenance is an importance part of multi-objective evolutionary algorithm. In this paper, a new variant for the NSGA-II algorithm is proposed. The basic idea is that using the crowding distance method designed by minimum spanning tree to maintain the distribution of solutions. From an extensive comparative study with NSGA-II on a number of two and three objective test problems, it is observed that the proposed algorithm has good performance in distribution, and is also rather competitive to NSGA-II concerning the convergence.

Keywords: Multi-objective optimization, Multi-objective evolutionary algorithm, Crowding distance, Minimum spanning tree.

1 Introduction

Pareto optimization [1, 2] has become a well established technique for detecting interesting solution candidates for multi-objective optimization problems (MOPs). It enables the decision maker to filter efficient solutions and to discover trade-offs between opposing objectives among theses solutions. In 1989, Goldberg [3] suggested the use of a genetic algorithm to solve MOPs and since then other investigators have been developing new methods, such as multi-objective genetic algorithm (MOGA) [4], niched Pareto genetic algorithm (NPGA) [5], non-dominated sorted genetic algorithm (NSGA [6] and NSGA-II [7]), strength Pareto evolutionary algorithm (SPEA [8] and SPEA2 [9]), Pareto archived evolution strategy (PAES [10]) and Memetic-PAES [11] etc..

Achieving a well-spread and well diverse Pareto solution front can be a time consuming computational problem, associated with multi-objective evolutionary algorithms (MOEAs). A good background review about the use of bounded archive population in MOEAs can be found in [12]. NSGA-II [7] uses a crowding distance approach which has a computational complexity of $O(MN\log N)$, where N is the population size and M is the objective number. SPEA [8] which has a computational complexity of $O(MN^3)$ had proved to be little better than NSGA-II for three objectives problem. Recently, several diversity maintenance methods have been studied, Kukkonen and Deb [13] proposed an improved pruning method based on crowding distance to truncate population, also the computational complexity of $O(MN\log N)$. It can provide good diversity in the case of two objectives, but when the number of

objectives is more than two, the obtained diversity declines drastically. On the other hand, some methods which achieve better results seem to have higher computational complexity. For example, SPEA2 [9] based on finding the k th nearest neighbor of solutions may have a computational complexity of $O(MN^2 \log N)$. MST-MOEA [14] maintained diversity by the MSTCD and degree of solution in minimum spanning tree could provide good extent and uniformity, however, the time complexity of the algorithm reaches $O(N^3)$. In addition, some methods based on a grid subdivided objective space [10, 15, 16] are proposed. The truncation approach of PAES and PESA2 is faster than that of SPEA2. However, some shortcomings can be found:

- a) The whole search space must be divided.
- b) The scale of each unit hypercube may be difficult to decide. If we choose an improper scale, we may get a poor distribution.

In this paper, an algorithm improving NSGA-II based on minimum spanning tree (MST) is proposed. We call it MST+NSGA-II. A minimum spanning tree for individuals set which has the largest rank is generated, and a 'first-generated' edge of individual is used to estimate the density (called improved crowding distance, ICD). The remainder of paper is organized as follows: Section 2 provides the original Crowding Distance method and its drawback in some cases. Section 3 describes the proposed method. Section 4 presents the algorithm settings, test functions and performance indices used for performance comparison. Section 5 shows the results and analysis of experiments carried out with MST+NSGA-II. Finally, some conclusions are discussed in Section 6.

2 Crowding Distance in NSGA-II

The population size becomes twice bigger than the original size at the end of generation of NSGA-II [7]. This bigger population is truncated based on the non-dominated sorting and Crowding Distance. Crowding Distance for a member of a non-dominated set tries to approximate the perimeter of a cuboid formed by using the nearest neighbors of the member. As indicated by Figure 1, the Crowding Distance of the individual i is the average side-length of the rectangle enclosed by dashed lines. And for boundary individuals (i.e., which have maximum or minimum value for any objective, e.g., individual j in Figure 1), Crowding Distance is assigned to have an infinite value. Finally, the individuals are sorted in monotonically decreasing order according to Crowding Distance and a desired number of individuals having the largest Crowding Distance values are selected.

Crowding Distance has a low computational complexity maintaining diversity, besides NSGA-II, it has been used as in many other MOEAs [17, 18, 19]. However, although the idea of this algorithm is reasonable, it does not provide good result in all cases (especially when the number of objectives is increased, the Crowding Distance does not work as well as expected [20]). Next, a main problem of Crowding Distance has been illustrated in Figure 2.

It is shown in Figure 2 that, individual B and C located very close to each other, but they are far from the other individuals. In Crowding Distance measurement, their value may be quite close, and it's very likely that they will both be removed or

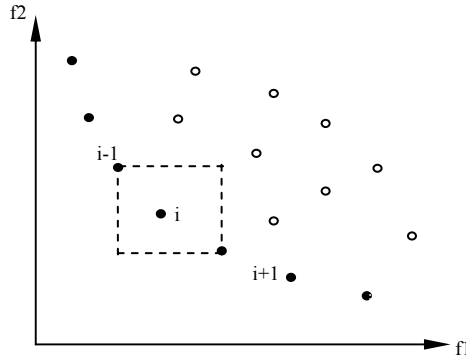


Fig. 1. The crowding distance calculation is shown

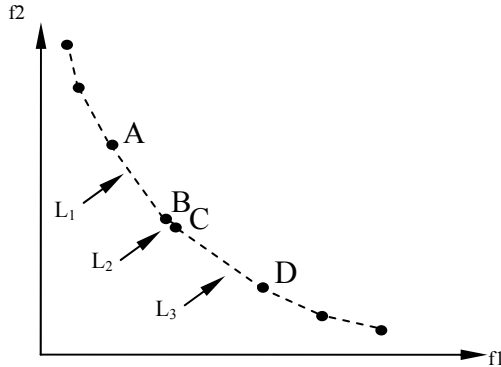


Fig. 2. An example of distribution

reserved. Obviously, it doesn't benefit the distribution of non-dominated set. The closer view suggests that either B or C should be removed in order to have a better spacing. Some diversity maintenance methods can solve this problem, such as SPEA2 [9] and MST-MOEA [14], however they need more Time consumption. An improved pruning algorithm based on Crowding Distance proposed by Kukkonen and Deb [13] provides good distribution for 2-objective problem. Nevertheless, it seems to not straighten out the difficulty in Figure 2.

3 Proposed Algorithm

To get a density estimation of solutions, initially, we calculate the Euclidean distance between any two solutions in the population. Then generate a minimum spanning tree for whole population, and the ICD of solution is a corresponding edge having this solution as a node in MST. It needs to point out that each edge can only represent the ICD of exclusive solution. Note that the number of edges of MST is $N - 1$, the initial

edge (i.e. the edge generated firstly by MST algorithm) represents the ICD of two individuals. The algorithm is described as following.

Algorithm 1

```

an improved-crowding-distance-assignment (I)
  for each j from 0 to N-1
    for each k from 0 to N-1
      {if( j != k)
         $C[j][k] = \text{Euclidean}(j, k)$  } // calculate the distance between
                                          any two solutions in I
  for each i from 0 to N-1,  $I[i]_{\text{distance}} = 0$  // initialize the ICD
  Prim_ICD (I) // generate MST using
‘Prim’ algorithm for N solutions // in I and assign the ICD for popula-
tion
  { for each i from 0 to N-1,  $a[i] = 0$  //  $a[i]$  is a mark of assignment,  $a[i] =$ 
0 represents // no-assignment,  $a[i] = 1$  represents
  already-assignment
     $a[0] = 1$ 
    findmin() // find the h, where  $C[0][h]$ 
is the minimum and  $a[h]=0$ 
     $a[h] = 1$ 
     $I[0]_{\text{distance}} = C[0][h]$ 
     $I[h]_{\text{distance}} = C[0][h]$  // the initial edge represents the
ICD of two solutions
    for i = 2 to N-1
      {findmin() // find the j and k, where
 $C[j][k]$  is the minimum and //  $a[j] = 1, a[k] = 0$ 
      assignment solutions // put k into already-
       $I[k]_{\text{distance}} = C[j][k]$  // assign the ICD of k
    }

```

In Algorithm 1, we design the ICD of solution by Prim algorithm. The ICD of solution is a ‘first-generated’ edge which connects this solution in MST. Therefore, the ICD of solutions B and C shown in Figure 2 are L_1 and L_2 or L_2 and L_3 respectively. Obviously, the ICD of B and C are more efficient than Crowding Distance in NSGA-II, considering that either B or C would be preserved only. In addition, for boundary solutions, as same as the strategy in NSGA-II, the ICD is assigned to have an infinite value.

The complexity of this procedure is governed by the calculation of the distance between any two solutions in population and generating the MST. Note that the time consumption of calculating distance of at most *N* solutions is $O(MN^2)$ and computational complexity of generating MST for population by ‘Prim’ algorithm is $O(N^2)$, the above algorithm has $O(MN^2) + O(N^2) = O(MN^2)$ computational complexity, which is worse than the $O(MN \log N)$ computational complexity designed by NSGA-II.

Table 1. The parameters of the experiments

Test problem	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6	DTLZ1	DTLZ2	DTLZ3
Population size	100	100	100	100	100	200	200	200
Evaluation number	20000	20000	20000	20000	20000	100000	100000	100000

4 Test Problems and Performance Indices

In order to validate the proposed method and quantitatively compare its performance with original NSGA-II, eight benchmark functions are tested by NSGA-II and MST+NSGA-II in this comparison study. And similarly to the NSGA-II, the proposed method is based on non-dominated fronts [7].

4.1 Genetic Parameter Settings

NSGA-II and MST+NSGA-II are given real-valued decision variables. A crossover probability of $p_c = 0.8$ and a mutation probability $p_m = 1/m$ (where m is the number of decision variables) are used. The operators for crossover and mutation are simulated binary crossover (SBX) and polynomial mutation with distribution indexes of $\eta_c = 11$, and $\eta_m = 17$ [17], respectively.

4.2 Functions Used

The test bed is formed by a total of eight functions. They are ZDT1, ZDT2, ZDT3, ZDT4, ZDT6 [22], DTLZ1, DTLZ2, DTLZ3 [21]. The number of decision variables m : $m_{ZDT1} = m_{ZDT2} = m_{ZDT3} = 30$, $m_{ZDT4} = m_{ZDT6} = 10$, $m_{DTLZ2} = m_{DTLZ3} = 12$, $m_{DTLZ1} = 7$. ZDT1, ZDT2, ZDT3, ZDT4, ZDT6 have 2 objectives and DTLZ1, DTLZ2, DTLZ3 have 3 ones. The evaluation number and population size of eight problems are given in Table 1.

4.3 Performance Metrics

Zitzler et al [22] suggested three goals that MOEAs can be identified and measured:

1. The distance of the resulting solutions to Pareto front (PF) should be minimized.
2. A uniform distribution of the solutions found is desirable.
3. The extent of the solutions should be maximized.

In this paper, five metrics (GD [23], SP [24], UA [25], D [22], IGD [26]) are used to assess the performance of the algorithms. GD assesses the algorithm on the first goal, SP and UA appraise the algorithm on the second goal, D scores the algorithm on the third goal and IGD is a comprehensive evaluation on three goals. Next, we introduce these metrics briefly.

GD [23] represents the average distance of the solutions in an approximation to PF. SP [24] and UA [25] both assess the uniformity of obtained non-dominated front. SP measures the standard deviation of distances from each solution to nearest solution in

the obtained non-dominated front. A smaller value of SP is better and for an ideal distribution $SP = 0$. However, the result of SP influenced by the size of solutions, convergence etc. may be unreliable [25]. UA used to estimate the uniformity of obtained non-dominated front overcomes these deficiencies of SP , the detailed difference with SP described in [25]. This method takes the value between zero and one, the larger value is better (one is the ideal result). The maximum spread (D) [22] measures the length of the diagonal of a minimal hyper-box, which encloses the obtained non-dominated set, and a larger value tells about a larger extent between extreme solutions. IGD [26] is a comprehensive method to measure the performance of algorithms. It contains the obtained solutions to PF, the uniformity and extent of the obtained solutions. It must be pointed out that IGD and GD need to know PF. Since we are dealing with artificial test problems, the Pareto front is not hard to obtain. In our experiments we use 500 uniformly spaced Pareto-optimal solutions as the approximation of PF.

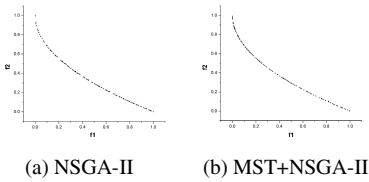


Fig. 3. The final solutions obtained by two MOEAs on ZDT1

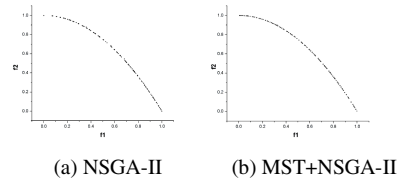


Fig. 4. The final solutions obtained by two MOEAs on ZDT2

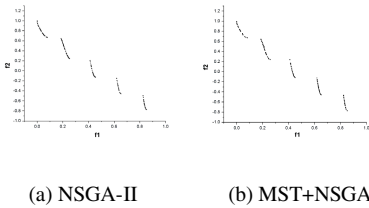


Fig. 5. The final solutions obtained by two MOEAs on ZDT3

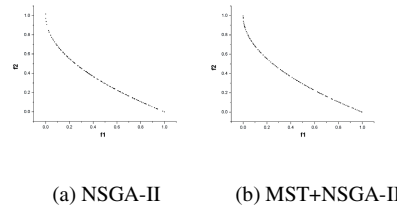


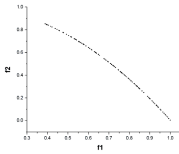
Fig. 6. The final solutions obtained by two MOEAs on ZDT4

Table 2. SP comparison of the two MOEAs. Better metric values are shown bold.

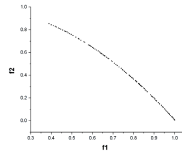
Algorithm	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6	DTLZ1	DTLZ2	DTLZ3
NSGA-II	0.00780	0.00770	0.00839	0.00775	0.00634	0.01814	0.03918	0.08262
	0.00078	0.00150	0.00040	0.00158	0.00210	0.00420	0.00189	0.05536
MST+NSGA-II	0.00496	0.00520	0.00611	0.00514	0.00363	0.00988	0.02305	0.02551
	0.00022	0.00059	0.00046	0.00123	0.00051	0.00447	0.00110	0.00671

Table 3. *UA* comparison of the two MOEAs. Better metric values are shown bold.

Algorithm	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6	DTLZ1	DTLZ2	DTLZ3
NSGA-II	0.40018	0.43200	0.42518	0.42284	0.30817	0.35312	0.34128	0.36580
	0.02217	0.02478	0.04158	0.03697	0.11650	0.02075	0.02365	0.02297
MST+NSGA-II	0.56379	0.55474	0.54098	0.53330	0.40837	0.65189	0.66691	0.68881
	0.02091	0.02937	0.03284	0.04801	0.13641	0.03275	0.02016	0.01702

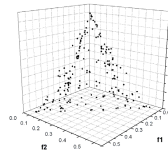


(a) NSGA-II

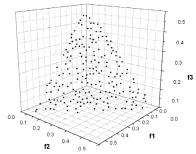


(b) MST+NSGA-II

Fig. 7. The final solutions obtained by two MOEAs on ZDT6



(a) NSGA-II



(b) MST+NSGA-II

Fig. 8. The final solutions obtained by two MOEAs on DTLZ1

5 Simulation Results

To compare the MST+NSGA-II and NSGA-II, we have carried out 20 independent runs, and the Table 2 - 6 includes the average and standard deviation, and the upper values in each row of the following table are the average values.

We analyze first the distribution of two MOEAs. From Figure 3 - 10, it is observed that the solutions obtained by MST+NSGA-II have better uniformity. Especially, it shows much obviously in three-objective problems. Table 2 and Table 3 show the results on *SP* and *UA*. In 2 - objective problems, MST+NSGA-II have a slight better uniformity results than NSGA-II. However, for 3 - objective problems, the advantage of MST+NSGA-II become more obvious, *SP* and *UA* suggest that the results obtained by MST+NSGA-II are almost two times better compared to those by NSGA-II. The extent of solution measured by *D* is shown in Table 4. For 2 - objective problems, it can be observed two methods have similar values. In ZDT2, ZDT4 and ZDT6, MST+NSGA-II has better results, and NSGA-II is better value in remaining problems. Nevertheless for 3 - objective problems, MST+NSGA-II have slightly better results than NSGA-II.

Table 4. *D* comparison of the two MOEAs. Better metric values are shown bold.

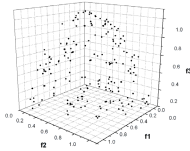
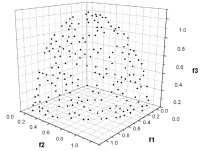
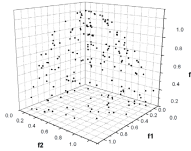
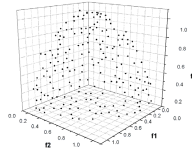
Algorithm	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6	DTLZ1	DTLZ2	DTLZ3
NSGA-II	1.4132	1.4141	1.9761	1.4142	1.0504	0.8613	1.7749	1.8363
	0.00064	0.00014	0.01927	8.5E-05	0.00949	0.04669	0.01283	0.82925
MST+NSGA-II	1.4053	1.4160	1.9524	1.4944	1.0511	0.8637	1.7767	1.9267
	0.00316	0.00405	0.00893	0.05840	0.00099	0.00964	0.00719	0.25295

Table 5. *GD* comparison of the two MOEAs. Better metric values are shown bold

Algorithm	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6	DTLZ1	DTLZ2	DTLZ3
NSGA-II	0.00032	0.00042	0.00059	0.00297	0.00089	1.4E-05	0.00011	0.00409
	0.00012	0.00018	4.1E-05	0.02373	0.00043	6.4E-06	9.4E-05	0.00457
MST+NSGA-II	0.00043	0.00030	0.00068	0.00186	0.00106	6.9E-05	2.7E-05	0.00296
II	0.00016	0.00014	4.6E-05	0.01974	0.00045	2.3E-05	1.9E-05	0.00255

Next, we analyze the convergence of two MOEAs. From Table 5, *GD* suggests that two methods have similar convergence results. In ZDT1, ZDT3, ZDT6 and DTLZ1, NSGA-II has better values, and MST+NSGA-II is better in remaining problems. In a word, this study shows that ICD, instead of Crowding Distance in NSGA-II, does not break the convergence of algorithm. Finally, we give a general comparison of two methods using *IGD*. In Table 6, MST+NSGA-II has the better result than NSGA-II in all problems except ZDT6 which illustrates that MST+NSGA-II has the better comprehensive performance than NSGA-II for almost test problems.

From above study, we find the advantage of MST+NSGA-II is more evident in three-objective problem. Considering our algorithm becomes more effective with the increasing of objective number, this sounds quite promising.

**Fig. 9.** The final solutions obtained by two MOEAs on DTLZ2**Fig. 10.** The final solutions obtained by two MOEAs on DTLZ3**Table 6.** *IGD* comparison of the two MOEAs. Better metric values are shown bold.

Algorithm	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6	DTLZ1	DTLZ2	DTLZ3
NSGA-II	0.00625	0.00686	0.00731	0.02069	0.00972	0.01766	0.04649	0.04786
	0.00095	0.00118	0.00107	0.00828	0.00868	0.00047	0.00267	0.00137
MST+NSGA-II	0.00619	0.00628	0.00709	0.01567	0.01122	0.01431	0.04105	0.04259
II	0.00150	0.00153	0.00095	0.00652	0.00556	0.00279	0.00181	0.00244

6 Conclusion

An improved crowding distance strategy based on minimum spanning tree to select non-dominated solutions which complete the new population was proposed. The new algorithm was integrated in NSGA-II in order to test its capacity to generate well distributed solutions. This technique was deployed in well known test functions and the results compared with the ones obtained by the NSGA-II Crowding Distance

method. The results show that our algorithm can reach a good diversity solution set, and it is also rather competitive to NSGA-II regarding the convergence.

It needs to point out that the proposed algorithm is more demanding in terms of computational load. However, the improvement gained in terms of diversity compensates this cost.

References

1. Coello, C.A.C., Van Veldhuizen, D.A., Lamont, G.B.: *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York (2002)
2. Deb, K.: *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, Chichester (2001)
3. Goldberg, D.E.: *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Reading (1989)
4. Fonseca, C.M., Fleming, P.J.: An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation* 3(1), 1–16 (1995)
5. Horn, J., Nafpliotis, N., Goldberg, D.E., Pareto, A.N.: Genetic Algorithm for Multiobjective Optimization. In: *Proceeding of the first IEEE Congress on Evolutionary Computation (CEC 1994)*, pp. 82–87 (1994)
6. Srinivas, N., Deb, K.: Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation* 2(3), 221–248 (1994)
7. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6(2), 182–197 (2002)
8. Zitzler, E., Thiele, L.: An evolutionary algorithm for Multiobjective optimization: The strength Pareto approach. Technical Report 43, Gloriastrasse 35, CH-8092 Zurich, Switzerland (1998)
9. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the strength Pareto evolutionary algorithm. *TIK-Report 103* (2001)
10. Knowles, J., Corne, D.: Approximating the nondominated front using the Pareto archive evolutionary strategy. *Evolutionary Computation* 8(2), 149–172 (2000)
11. Knowles, J., Corne, D.: M-PAES: A Memetic Algorithm for Multiobjective Optimization. In: *Proceeding of IEEE Congress on Evolutionary Computation (CEC 2000)*, pp. 325–332 (2000)
12. Knowles, J.D., Corne, D.W., Fleischer, M.: Bounded Archiving using the Lebesgue Measure. In: *Proceedings of IEEE Congress on Evolutionary Computation (CEC 2003)*, pp. 2490–2497 (2003)
13. Kukkonen, S., Deb, K.: Improved Pruning of Non-Dominated Solutions Based on Crowding Distance for Bi-Objective Optimization Problems. In: *Proceedings of IEEE on Evolutionary Computation (CEC 2006)*, pp. 3995–4002 (2006)
14. Li, M., Zheng, J., Xiao, G.: An Efficient Multi-objective Evolutionary Algorithm Based on Minimum Spanning Tree. In: *Proceedings of IEEE Congress on Evolutionary Computation (CEC 2008)*, pp. 617–624 (2008)
15. Corne, D.W., Jerram, N.R., Knowles, J.D., Oates, M.J.: PESA-II: Region-based Selection in Evolutionary Multiobjective Optimization. In: Spector, L., Goodman, E., Wu, A., Langdon, W.B., Voigt, H.-M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M.H., Burke, E. (eds.) *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, pp. 283–290. Morgan Kaufmann Publishers, San Francisco (2001)

16. Knowles, J., Corne, D.: Properties of an Adaptive Archiving Algorithm for Storing Non-dominated Vectors. *IEEE Transactions on Evolutionary Computation* 7(2), 100–116 (2003)
17. Aguirre, H., Tanaka, K.: Selection, drift, recombination, and mutation in multiobjective evolutionary algorithms on scalable mnk-landscapes. In: *Proceedings of the 3rd International Conference on Evolutionary Multi-Criterion Optimization (EMO 2005)*, Guanajuato, Mexico, pp. 355–369 (March 2005)
18. Kukkonen, S., Lampinen, J.: GDE3: The third evolution step of Generalized Differential Evolution. In: *Proceedings of IEEE Congress on Evolutionary Computation (CEC 2005)*, pp. 443–450 (2005)
19. Raquel, C.R., Naval Jr., P.C.: An effective use of crowding distance in multiobjective particle swarm optimization. In: *Proceedings of the Genetic and Evolutionary Computation (GECCO 2005)*, Washington, DC, USA, June, pp. 257–264 (June 2005)
20. Coello, C.A.C.: 20 Years of Evolutionary Multi-Objective Optimization: What Has Been Done and What Remains to be Done. In: Yen, G.Y., Fogel, D.B. (eds.) *Computational Intelligence: Principles and Practice*, ch. 4, pp. 73–88. *IEEE Computational Intelligence Society* (2006)
21. Deb, K., Thiele, L., Laumanns, M., Zitzler, E.: Scalable Test Problems for Evolutionary Multiobjective Optimization. In: Abraham, A., Jain, L., Goldberg, R. (eds.) *Evolutionary Multiobjective Optimization. Theoretical Advances and Applications*, pp. 105–145 (2005)
22. Zitzler, E., Deb, K., Thiele, L.: Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation* 8(2), 173–195 (2000)
23. Van Veldhuizen, D.A., Lamont, G.B.: Evolutionary Computation and Convergence to a Pareto Front. In: Koza, J.R. (ed.) *Late Breaking Papers at the Genetic Programming Conference*, pp. 221–228 (1998)
24. Schott, J.R.: Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization. Master's thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology (1995)
25. Li, M., Zheng, J., Xiao, G.: Uniformity Assessment for Evolutionary Multi-Objective Optimization. In: *Proceedings of IEEE Congress on Evolutionary Computation (CEC 2008)*, pp. 625–632 (2008)
26. Sierra, M.R., Coello, C.A.C.: A Study of Fitness Inheritance and Approximation Techniques for Multi-Objective Particle Swarm Optimization. In: *Proceedings of IEEE Congress on Evolutionary Computation (CEC 2005)*, pp. 65–72 (2005)

An Island Based Hybrid Evolutionary Algorithm for Optimization

Changhe Li and Shengxiang Yang

Department of Computer Science, University of Leicester
University Road, Leicester LE1 7RH, UK
{c1160,s.yang}@mcs.le.ac.uk

Abstract. Evolutionary computation has become an important problem solving methodology among the set of search and optimization techniques. Recently, more and more different evolutionary techniques have been developed, especially hybrid evolutionary algorithms. This paper proposes an island based hybrid evolutionary algorithm (IHEA) for optimization, which is based on Particle swarm optimization (PSO), Fast Evolutionary Programming (FEP), and Estimation of Distribution Algorithm (EDA). Within IHEA, an island model is designed to cooperatively search for the global optima in search space. By combining the strengths of the three component algorithms, IHEA greatly improves the optimization performance of the three basic algorithms. Experimental results demonstrate that IHEA outperforms all the three component algorithms on the test problems.

1 Introduction

Evolutionary computation has become an important search and optimization technique for many researchers. The population-based parallel computation, collective learning process, self-adaptation, and robustness are some of the key features of evolutionary algorithms (EAs). EAs have been widely applied for solving important practical problems in engineering, business, and commerce, etc., yet in practice sometimes they deliver only marginal performance. There is little reason to expect that one can find a uniformly optimal algorithm for solving all optimization problems. This is in accordance with the No Free Lunch theorem [1], which explains that for any algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class. Recently, the hybridization of EAs is becoming more and more popular due to their capabilities in handling several real world problems that involve complexity, noisy environments, imprecision, uncertainty, and vagueness.

As reported in the literature, several heuristical techniques have been used to improve the general efficiency of EAs. Zmuda et al. [2] introduced a hybrid evolutionary learning scheme for synthesizing multi-class pattern recognition systems. Wang [3] developed a hybrid approach to improve the performance of EAs for a simulation optimization problem. A hybrid evolutionary Particle Swarm Optimization (PSO) method was proposed by Shi et al. [4]. The hybrid approach

executes two systems simultaneously and selects P individuals from each system for exchanging after the designated N iterations. The individuals with a larger fitness have more opportunities of being selected. A hybrid technique that combines GA and PSO, called genetic swarm optimization (GSO), was proposed by Grimaldi et al. [5] for solving an electromagnetic optimization problem. Li and Wang et al. [6,7] proposed a hybrid PSO using Cauchy mutation to reduce the probability of trapping local optima for PSO.

In this paper, an island based hybrid evolutionary algorithm(IHEA) is proposed based on Particle Swarm Optimization (PSO), Fast Evolutionary Programming (FEP) and Estimation of Distribution Algorithm (EDA). An island model using different evolutionary strategies is designed for improving the optimization performance of the component algorithms. A set of function optimization benchmark problems are tested in this paper.

The rest of the paper is organized as follows. Section 2 briefly describes PSO, FEP, and EDA optimization techniques. The proposed algorithm, IHEA, is presented in details in Section 3. Experimental results are shown in Section 4. Finally, some conclusions are given in Section 5.

2 Three Different Evolutionary Techniques

2.1 Particle Swarm Optimization

Particle swarm optimization (PSO) was first introduced by Kennedy and Eberhart in 1995 [8,9]. PSO is motivated from the social behavior of organisms, such as bird flocking and fish schooling. Particles “fly” through the search space by following the previous best positions of their neighbors and their own previous best positions. There are several main versions of the PSO algorithms. The following version modified by Shi and Eberhart [10] is used in this paper. Each particle is represented by a position and a velocity, which are updated as follows:

$$\mathbf{V}'_i = \omega \mathbf{V}_i + \eta_1 r_1 (\mathbf{P}_i - \mathbf{X}_i) + \eta_2 r_2 (\mathbf{P}_g - \mathbf{X}_i) \quad (1)$$

$$\mathbf{X}'_i = \mathbf{X}_i + \mathbf{V}'_i \quad (2)$$

where $\mathbf{X}'_i$ and $\mathbf{X}$ represent the current and previous positions of particle i , $\mathbf{V}_i$ and $\mathbf{V}'_i$ are the previous and current velocity of particle i , $\mathbf{P}_i$ and $\mathbf{P}_g$ are the best-so-far position of particle i and the best position found in the whole swarm so far respectively. $\omega \in (0, 1]$ is an inertia weight which determines how much the previous velocity is preserved, η_1 and η_2 are acceleration constants, and r_1 and r_2 are random numbers generated from the interval $[0.0, 1.0]$.

The framework of the PSO algorithm is given as follows:

- Step 1: Generate the initial particles by randomly generating the position and velocity for each particle.
- Step 2: Evaluate the fitness of each particle.
- Step 3: For each particle i , if its fitness is smaller than the fitness of its previous best position ($\mathbf{P}_i$), update $\mathbf{P}_i$.

- Step 4: For each particle, if its fitness is smaller than the fitness of the best position ($\mathbf{p}_g$) of all particles, update $\mathbf{P}_g$.
 Step 5: Update each particle according to Eqs. (1) and (2).
 Step 6: Stop if the stop criterion is satisfied; otherwise, go to Step 3.

From the mathematic theoretical analysis of the trajectory of a PSO particle [11], the trajectory of a particle $\mathbf{X}_i$ converges to a weighted mean of $\mathbf{P}_i$ and $\mathbf{P}_g$. Whenever the particle converges, it will “fly” to the personal best position and the global best particle’s position. This information sharing mechanism makes PSO have a very fast speed of convergence. Meanwhile, because of this mechanism, PSO can’t guarantee to find the global minimal value of a function.

2.2 Fast Evolutionary Programming

Fast Evolutionary Programming (FEP) was proposed by Yao [12] by introducing Cauchy mutation instead of Gaussian mutation in classical EP (CEP). The framework of FEP is as follows:

- Step 1. Generate initial population of μ individuals. Each individual is represented by a pair of real-valued vectors $(x_i, \eta_i), \forall i \in \{1, 2, \dots, \mu\}$.
 Step 2. Evaluate the fitness of each individual (x_i, η_i) .
 Step 3. For each parent $(x_i, \eta_i), \forall i \in \{1, 2, \dots, \mu\}$, create a single offspring $(x'_i, \eta'_i), \forall i \in \{1, 2, \dots, \mu\}$ by:

$$x'_i(j) = x_i(j) + \eta_i(j)\delta_j \quad (3)$$

$$\eta'_i(j) = \eta_i(j)\exp(\tau'N(0, 1) + \tau N_j(0, 1)) \quad (4)$$

where $x_i(j)$, $x'_i(j)$, $\eta_i(j)$, and $\eta'_i(j)$ denote the j -th component of individual (x_i, η_i) respectively. $N(0, 1)$ denotes a normally distributed one-dimensional random number with mean zero and standard deviation one. $N_j(0, 1)$ indicates that the random number is generated anew for each value of j . δ_j is a Cauchy random variable with the scale parameter 1 and is generated anew for each value of j . The factor τ and τ' are usually set to $(\sqrt{2\sqrt{n}})^{-1}$ and $\sqrt{2n}^{-1}$ [13,14].

- Step 4. Calculate the fitness of each offspring $(x'_i, \eta'_i), \forall i \in \{1, 2, \dots, \mu\}$.
 Step 5. Conduct pairwise comparison over the union of parents (x'_i, η'_i) and offspring $(x'_i, \eta'_i), \forall i \in \{1, 2, \dots, \mu\}$. For each individual, q opponents are chosen uniformly at random from all the parents and offspring. For each comparison, if the individual’s fitness is no smaller than the opponent’s, it receives a “win”.
 Step 6. Select μ individuals from the parents and offspring that have the most wins to next generation.
 Step 7. If stop criterion is satisfied, then stop; otherwise, go to Step 3.

Yao et al. [15] explains that Cauchy mutation performs better than Gaussian mutation for most tested benchmark problems used in [15] because of its higher

probability of making longer jumps (the larger the search step size, the faster the algorithm gets to the global optimum). However, the problem of FEP is that a large step size may not be beneficial at all if the current search point is already very close to the global optimum.

2.3 Estimation of Distribution Algorithms

Estimation of Distribution Algorithms (EDAs) [16,17] are non-deterministic, stochastic heuristic search strategies that form part of the evolutionary computation approaches. Within EDAs, a number of solutions or individuals are created every generation according to a distribution model, which evolves generation by generation until a satisfactory solution is achieved. In brief, the characteristic that most differentiates EDAs from other evolutionary search strategies, such as genetic algorithms, is that the evolution from a generation to the next one is done by estimating the probability distribution of the fittest individuals, and afterwards by sampling the induced model. This avoids the use of crossover or mutation operators, and the number of parameters that EDAs require is reduced considerably.

The framework of EDAs is as follows:

- Step 1. Select M promising individuals from parent population to form the parent set Q by a selection method (e.g., the truncation selection).
- Step 2. Build a probabilistic model $p(x)$ based on the statistical information extracted from the parent set Q .
- Step 3. Sample offspring according to the constructed probability model $p(x)$.
- Step 4. Fully or partly replace individuals in parent population by sampled offspring to form the next generation.

In this paper, we use truncation selection method as most literatures used. Another major issue in EDA is how to build a probability distribution model $p(x)$. we use Gaussian model with diagonal covariance matrix (GM/DCM) [17]. In GM/DCM, the joint density function of the k -th generation is described as follows:

$$p_k(x) = \prod_{i=1}^n \mathcal{N}(x_i; \mu_i^k, \sigma_i^k) \quad (5)$$

where

$$\mathcal{N}(x_i; \mu_i^k, \sigma_i^k) = \frac{1}{\sqrt{2\pi\sigma_i^k}} e^{-\frac{1}{2}(\frac{x_i - \mu_i^k}{\sigma_i^k})^2} \quad (6)$$

In Equ(5), the n -dimensional joint probability distribution is factorized as a product of n univariate and independent normal distributions. There are two parameters for each variable required to be estimated in the k -th generation: the mean $\hat{\mu}_i^k$, and the standard deviation, $\hat{\sigma}_i^k$. They can be estimated as follows:

$$\hat{\mu}_i^k = \bar{X}_i^k = \frac{1}{M} \sum_{t=1}^M x_{t,i}^k; \hat{\sigma}_i^k = \sqrt{\frac{1}{M} \sum_{t=1}^M (x_{t,i}^k - \bar{X}_i^k)^2} \quad (7)$$

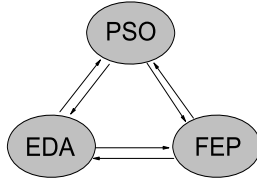


Fig. 1. Migration topology of IHEA

where $(x_{1,i}^k, x_{2,i}^k, \dots, x_{M,i}^k)$ are values of the i -th variable of the selected M parent individuals in the k -th generation.

Experimental results show that EDA gives distinctive performance on some particular problems. However, it doesn't help for the search on some other problems at all. In EDA, the search is mainly based on global information used for guiding the exploration. However, too much exploration may cause no convergence problem.

3 Island Based Hybrid Evolutionary Algorithm

The main idea of IHEA is that migration of individuals among different islands can increase diversity of each island, so it can reduce the probability of premature convergence. In IHEA, there are three subpopulation residing in three different islands, which use PSO, FEP and EDA algorithms respectively to search global optima in the whole shared search space. Since different islands use different evolutionary techniques, they probably follow different search directions in the whole shared search space, that is they explore different area in the whole search space. However, they are not independent to search, they exchange their own updated information periodically between each other by migration of promising individuals. The information sharing mechanism is helpful to search unexplored space where probably global optima is.

PSO in IHEA can be regarded as a fast local search operator for exploitation, EDA is used for exploring new promising area in the whole search space. FEP can be taken as mutation operator because of its long jump capability. The cooperative search among the three islands is helpful for them to explore new promising areas. It greatly reduces the probability of premature convergence, hence the global search capability is improved.

The migration topology of IHEA is described in Fig. 1. The main framework of IHEA is as follows:

- Step 1. Initialize the three population, set $k = 1$.
- Step 2. Use PSO, EDA and FEP to optimize each population.
- Step 3. Compare the best individual of each population, if the best individual of population p is better than the best of the other two population, migrate individuals of population p to the other two population whose fitness is better than those individuals of the other two population.
- Step 4. Stop if stop criterion is true; otherwise, $k = k + 1$ and goto Step 2.

Table 1. Details of the test functions, where n and f_{min} are the dimension and the minimum value of a function respectively and $S \in R_n$

Test Function	n	S	f_{min}
$f_1(x) = \sum_{i=1}^n x_i^2$	30	$(-5.12, 5.12)$	0
$f_2(x) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2$	2	$(-2.048, 2.048)$	0
$f_3(x) = \frac{1}{4000} \sum_{i=1}^n (x_i - 100)^2 - \prod_{i=1}^n \cos(\frac{x_i - 100}{\sqrt{i}}) + 1$	30	$(-300, 300)$	0
$f_4(x) = -20 \exp(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2}) - \exp(\frac{1}{n} \sum_{i=1}^n \cos(2\pi x_i)) + 20 + e$	30	$(-30, 30)$	0
$f_5(x) = \sum_{i=1}^n 100(x_{i+1}^2 - x_i)^2 + (x_i - 1)^2$	30	$(-2.048, 2.048)$	0
$f_6(x) = \sum_{i=1}^n (x_i^2 - 10 \cos(2\pi x_i) + 10)$	30	$(-5.12, 5.12)$	0
$f_7(x) = \sum_{i=1}^n -x_i \sin(\sqrt{ x_i })$	30	$(-500, 500)$	-12569.5
$f_8(x) = 418.9829 \cdot n + \sum_{i=1}^n -x_i \sin(\sqrt{ x_i })$	30	$(-500, 500)$	0

Table 2. Population size of the three islands in IHEA algorithm

Island	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
PSO	10	15	10	10	15	5	10	10
FEP(q)	5(3)	10(4)	15(5)	5(3)	10(4)	10(4)	15(5)	15(5)
EDA	15	5	5	15	5	15	5	5

Another big issue in IHEA is the population resource allocation among different islands. As we know, different problems may have totally different landscapes, such as the number of local or global optima, the difference among local optima, the location of local or global optima, and so on. Different problems may need different algorithms to solve it. This can be observed from the result presented in Section 4.2. Hence, we should allocate different population resources on different islands. For example, we can allocate the most population resources to the island that is more effective than other algorithms to solve the problem.

4 Experimental Study

4.1 Test Problems and Experimental Settings

Eight benchmark functions ($f_1 - f_8$) are used in this paper. Functions $f_1 - f_2$ is unimodal functions while $f_3 - f_8$ have many local optima. Table 1 gives the details of these functions. Algorithm parameters are set as follows: For PSO algorithm, the acceleration constants of η_1 and η_2 are both set to be 1.496180 and the inertia weight $\omega = 0.729844$ as suggested by den Bergh [18]. In FEP, the tournament size is 5 for selection and initial standard deviation is 3.0 as used in [12]. For all algorithms, population size is 30. The subpopulation size of FEP, EDA and PSO is different for different problem in IHEA algorithm, details can be seen from Table 2, however, the values of population size are just experimental values not optima values. we run 50 times independently till generation 1500 for all test problems.

Table 3. Comparison of PSO, EDA, FEP and IHEA. The results are mean best function values found at generation 1500 over 50 runs.

Test function	PSO	EDA	FEP	IHEA
f_1	1.218e-021	0	1.0219	0
f_2	0	0.3835	3.26304e-005	0
f_3	0.0291024	54.8026	1.29825	0.0559343
f_4	2.09482	4.44089e-016	4.77732	4.44089e-016
f_5	9.09728e-012	2545.73	80.2489	1.71309e-009
f_6	50.7826	0	26.0043	0
f_7	-7431.63	-2343.9	-12307	-12522.4
f_8	5043.08	10348.7	314.947	71.0184

Table 4. The T-test results between IHEA and the other three algorithms, where “***” means the results of each run is the same

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
PSO	-1.17951	***	3.13962	-12.9764	2.03565	-26.3966	-73.2857	-71.1316
EDA	***	-13.8161	-108.133	0	-250.172	***	-137.587	-183.923
FEP	-4.18079	-1.31905	-10.73	-14.7223	-9.02354	-21.3865	-6.90319	-7.33192

4.2 Performance Comparison

The average best results of the four algorithms are shown in Table 3. Table 4 shows the statistical comparison of the IHEA algorithm over the other three algorithms, using the two-tailed T-test with 98 degrees of freedom at a 0.05 level of significance. In all the T-test results of the paper, the performance difference is significant if the absolute value of the T-test result is greater than 1.984.

From Table 3 and Table 4, we can see that PSO, EDA and FEP give quite different performance on different test cases. PSO obtains the best results on function f_2 , f_3 and f_5 , EDA gives the best results on function f_1 , f_4 and f_6 , while FEP presents the best results on function f_7 and f_8 . However, when compare the three algorithms with IHEA, we can find that IHEA shows the best performance on all test problems except function f_3 and f_5 , which is slightly worse than the results of PSO. IHEA greatly improves the performance of the three original algorithms.

Fig 2 also shows the evolutionary process of the four algorithms. IHEA gives the fastest convergence on most test problems. From the results of Table 3 and Fig 2, we can conclude that IHEA reduces the probability of premature convergence for most tested problems, especially for multimodal functions.

Table 5 and Table 6 show the average results of average mean best and T-test results of all the test problems of 10 dimensions except function f_2 . Form the results, we can see that all the results obtained by IHEA are better than that of the other three algorithms except on function f_5 , where the result of IHEA is slightly worse than that of PSO.

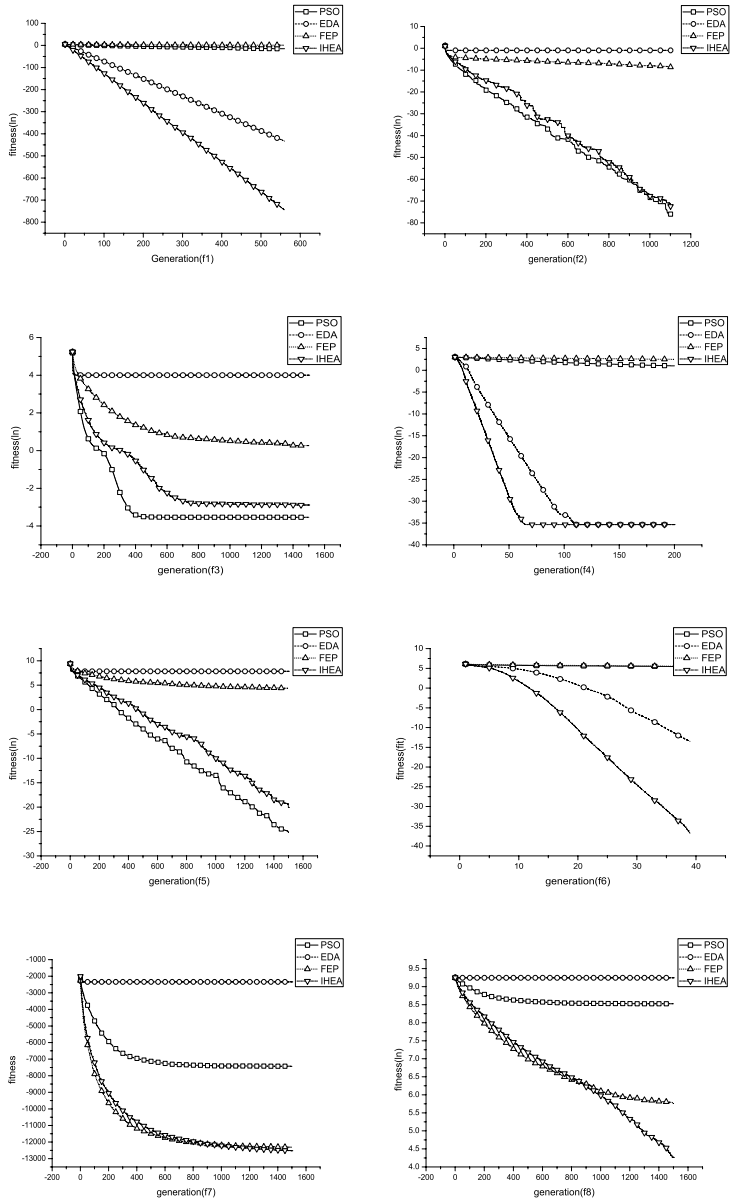


Fig. 2. Evolution process of the average best particle of PSO, EDA, FEP, and IHEA

Table 5. Comparison of PSO, EDA, FEP and IHEA on problems of 10 dimensions. The results are mean best function values found at generation 1500 over 50 runs.

Test function	PSO	EDA	FEP	IHEA
f_1	6.98296e-072	0	0.00744029	0
f_3	0.0866419	11.5354	0.18873	0.073003
f_4	4.06785e-015	4.44089e-016	0.575907	4.44089e-016
f_5	1.97215e-031	599.55	0.0918242	4.19082e-030
f_6	6.18864	0	6.41484	0
f_7	-3431.8	-1317.14	-4177.4	-4189.83
f_8	725.257	2881.37	10.2413	0.000127276

Table 6. The T-test results between IHEA and the other three algorithms on problems of 10 dimensions, where “***” means the results of each run is the same

	f_1	f_3	f_4	f_5	f_6	f_7	f_8
PSO	-1.90455	-1.84457	-50.9999	3.20954	-15.5783	-21.8944	-20.3932
EDA	***	-37.4252	0	-51.6912	***	-80.4453	-66.1889
FEP	-1.94113	-2.24096	-4.50941	-1.67656	-11.619	-2.54557	-1.97234

5 Conclusions

This paper discusses an effective method to reduce the probability of trapping local optima for evolutionary algorithm, which employs hybrid method to improve algorithm performance. An island based hybrid algorithm (IHEA) is proposed based on PSO, EDA and FEP algorithms. In IHEA, three different islands were designed using PSO, EDA and FEP as their local search method respectively. An information sharing mechanism is developed among the three islands. A set of benchmark problems were tested. From the result, it can be seen that IHEA greatly reduces the probability of becoming trapped at a local optimum. The global search capability of IHEA is better than the three original algorithms.

In the future, we will focus on the further study of IHEA in deep level. There are some issues that should be considered, such as, the resources distribution of different islands, the migration topology and algorithms used on different islands.

Acknowledgement

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of UK under Grant EP/E060722/1.

References

1. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.* 1(1), 67–82 (1997)
2. Zmuda, M.A., Rizki, M.M., Tamburino, L.A.: Hybrid evolutionary learning for synthesizing multi-class pattern recognition systems. *Applied Soft Computing* 2(4), 269–282 (2003)
3. Wang, L.: A hybrid genetic algorithm-neural network strategy for simulation optimization. *Applied Mathematics and Computation* 170(2), 1329–1343 (2005)
4. Shi, X.H., Liang, Y.C., Lee, H.P., Lu, C., Wang, L.M.: An improved GA and a novel PSO-GA-based hybrid algorithm. *Information Processing Letters* 93(5), 255–261 (2005)
5. Grimaldi, E.A., Grimacia, F., Mussetta, M., Pirinoli, P., Zich, R.E.: A new hybrid genetical C swarm algorithm for electromagnetic optimization. In: *Proc. of Int. Conf. on Computational Electromagnetics and its Applications*, pp. 157–160 (2004)
6. Li, C., Liu, Y., Kang, L., Zhou, A.: A Fast Particle Swarm Optimization Algorithm with Cauchy Mutation and Natural Selection Strategy. In: Kang, L., Liu, Y., Zeng, S. (eds.) *ISICA 2007. LNCS*, vol. 4683, pp. 334–343. Springer, Heidelberg (2007)
7. Wang, H., Liu, Y., Li, C., Zeng, S.: A Hybrid Particle Swarm Algorithm with Cauchy Mutation. In: *Proc. of the 2007 IEEE Swarm Intelligence Symposium* (2007)
8. Eberhart, R.C., Kennedy, J.: A new optimizer using particle swarm theory. In: *Proc. of the 6th Int. Symp. on Micro Machine and Human Science*, pp. 39–43 (1995)
9. Kennedy, J., Eberhart, R.C.: Particle Swarm Optimization. In: *Proc. of the 1995 IEEE Int. Conf. on Neural Networks*, pp. 1942–1948 (1995)
10. Shi, Y., Eberhart, R.C.: A Modified Particle Swarm Optimizer. In: *Proc. of the IEEE Int. Conf. on Evol. Comput.*, pp. 69–73 (1998)
11. Clerc, M., Kennedy, J.: The Particle Swarm: Explosion, Stability and Convergence in a Multi-Dimensional Complex Space. *IEEE Trans. on Evol. Comput.* 6(1), 58–73 (2002)
12. Yao, X., Liu, Y.: Fast evolutionary programming. In: *Proc. of the 5th Annual Conference on Evolutionary Programming (EP 1996)*, pp. 451–460 (1996)
13. Bäck, T., Schwefel, H.-P.: An overview of evolutionary algorithms for parameter optimization. *Evol. Comput.* 1(1), 1–23 (1993)
14. Fogel, D.B.: An introduction to simulated evolutionary optimization. *IEEE Trans. Neural Networks* 5(1), 3–14 (1994)
15. Yao, X., Liu, Y., Lin, G.: Evolutionary programming made faster. *IEEE Trans. on Evol. Comput.* 3(1), 82–102 (1999)
16. Zhang, B.-T.: A Bayesian framework for evolutionary computation. In: *Proc. of the 1999 Congress on Evol. Comput.*, pp. 722–728 (1999)
17. Larrañaga, P., Lozano, J.A.: *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, Dordrecht (2001)
18. van den Bergh, F.: *An Analysis of Particle Swarm Optimizers*. PhD thesis, Department of Computer Science, University of Pretoria, South Africa (2002)

A Particle Swarm Optimization Based Algorithm for Fuzzy Bilevel Decision Making with Objective-Shared Followers

Ya Gao, Guangquan Zhang, and Jie Lu

Faculty of Engineering and Information Technology, University of Technology,
Sydney PO Box 123, NSW 2007, Australia
{yagao,zhangg,jielu}@it.uts.edu.au

Abstract. A bilevel decision problem may have multiple followers as the lower decision units and have fuzzy demands simultaneously. This paper focuses on problems of fuzzy linear bilevel decision making with multiple followers who share a common objective but have different constraints (FBOSF). Based on the ranking relationship among fuzzy sets defined by cut set and satisfactory degree, a FBOSF model is presented and a particle swarm optimization based algorithm is developed.

Keywords: Bilevel programming, bilevel multiple follower decision making, particle swarm optimization, fuzzy sets.

1 Introduction

Bilevel decision problems, initialed by Von Stackelberg [1], are mainly developed for solving decentralized management problems with decision makers in a hierarchy. The upper is termed the leader and the lower the follower [2]. The investigation of bilevel problems is strongly motivated by real world applications, and bilevel decision support techniques have been applied with remarkable success in different domains such as decentralized resource planning [3], electronic power market [4], logistics [5], civil engineering [6], and road network management [7, 8].

A large part of the research on bilevel decision problems has centered on its linear version, for which there have proposed nearly two dozen algorithms [9, 10, 11, 12, 13, 14, 15]. These algorithms can be roughly classified into three categories: the vertex enumeration based approaches [9, 10], which use the important characteristic that at least one global optimal solution is attained at an extreme point of the constraints set; the Kuhn-Tucker approaches [11, 12, 13] in which a bilevel problem is transferred into a single level problem that solves the leader's problem while including the follower's optimality conditions as extra constraints; and the heuristics [14, 15, 16], which are known as global optimization techniques based on convergence analysis.

In cases of real-world bilevel decision making problems, certain practical issues are frequently confronted.

First, the lower level of a bilevel decision may involve more than one decision unit. The leader's choice is therefore affected by the objectives and strategies of his or her lower counterparts. For each possible decision from the leader, those followers may have their own different reactions. Multi-follower bilevel decision problems have been studied in our research lab [17]. A framework has been established and a total of 9 sub problems are identified according to different levels of cooperation among follower-controlled variables, objectives, and constraints respectively. This research focuses on the situation when followers have individual decision variables and constraints, but share the same objective.

Second, when modeling a real bilevel decision problem, it is sometimes difficult to determine parameters which define the problem. These parameters are often required to be fixed at some values in an experimental and/or subjective manner through the experts' understanding of the nature of the parameters in the problem-formulation process [18]. It has been observed that, in most real-world situations the possible values of these parameters are often only imprecisely or ambiguously known to the experts who establish this model. Therefore, fuzzy bilevel models are proposed, in which the parameters, either in objective functions or in constraints of the leader and/or the follower, are described by fuzzy values. The fuzzy BP problem was first studied by Sakawa et al. [19], who have formulated cooperative fuzzy bilevel decision problems and proposed a fuzzy bilevel programming approach to these problems. An approximation approach has been developed in our research team [20, 21, 22, 23] based on the bilevel optimization framework building and models formatting [17, 24]. Solutions can be reached by solving associated multiple objectives bilevel decision problem under different cut sets.

Since bilevel mathematical program is NP-hard due to its non-convexity and non-differentiability, it is difficult to find solution algorithms suited to this particularity [25, 26]. Existing solution algorithms or methods are still inefficient and lack universality [27]. Heuristic approaches are now generating interest in the research community as an alternative for solving the bilevel problems [28].

Particle swarm optimization (PSO) is a heuristic algorithm proposed by James Kennedy and Russell Eberhart in 1995 [29]. It is one of community intelligent algorithms for searching a global solution, which comes from the study of the simple model of bird community and bird behavior simulation [27]. As PSO "requires only primitive mathematical operators, and is computationally inexpensive in terms of both memory requirements and speed" [30, 31], it has a good convergence performance and has been successfully applied in many fields such as neural network training [32], integral programming [18] and multi-object optimization [33].

This paper applies PSO technique on FBOSF problems, and it is organized in 4 sections. Following the introduction in Section 1, Section 2 gives definitions and techniques used in subsequent sections. In Section 3, based on a FBOSF model, a PSO based algorithm is developed. Section 4 concludes this paper and highlights the future research.

2 Preliminaries

In this section, some definitions and techniques used in subsequent sections are presented.

Throughout this paper, $\mathbb{R}$ represents the set of all real numbers, $\mathbb{R}^n$ is n -dimensional Euclidean space, $F(\mathbb{R})$ and $F^n(\mathbb{R})$ are the set of all fuzzy numbers and the set of all n -dimensional fuzzy numbers on $\mathbb{R}^n$ respectively.

2.1 A Fuzzy Ranking Method

Definition 1. [11] For any n -dimensional fuzzy vectors $\tilde{a} = (\tilde{a}_1, \dots, \tilde{a}_n)$, $\tilde{b} = (\tilde{b}_1, \dots, \tilde{b}_n)$, $\tilde{a}_i, \tilde{b}_i \in F(\mathbb{R})$, under a certain satisfactory degree $\alpha \in [0, 1]$, we define

$$\tilde{a} \preceq_{\alpha} \tilde{b} \text{ iff } a_{i\lambda}^L \leq b_{i\lambda}^L \text{ and } a_{i\lambda}^R \leq b_{i\lambda}^R, i = 1, 2, \dots, n, \forall \lambda \in [\alpha, 1]$$

where α is the adjustable satisfactory degree, which means, when comparing two fuzzy numbers all values with membership grades smaller than α are neglected. $\tilde{a}_{i\lambda}^L$ and $\tilde{a}_{i\lambda}^R$ are the left and right λ -cut of $\tilde{a}_i$; $\tilde{b}_{i\lambda}^L, \tilde{b}_{i\lambda}^R$ are defined the same way.

2.2 Particle Swarm Optimization

Inspired by the social behavior of animals such as fish schooling and bird flocking, PSO is a kind of population-based algorithm. The population of PSO is called “swarm”, and each individual in the swarm is called “particle”. The similarity between PSO and other evolutionary algorithms lies in that the individual in the community is moved to a good area according to the fitness to the environment. Unlike other evolutionary computation methods, however, each particle in PSO has an adaptable velocity (position change), according to which it moves in the search space [31]. Moreover, each particle has a memory, remembering the best position of the search space it has ever visited [30]. Thus, its movement is an aggregated acceleration towards its best previously visited position and towards the best particle of a topological neighborhood.

Suppose current search space for PSO is n -dimensional, then the i^{th} particle of the swarm can be represented by a n -dimensional vector, $x_i = (x_{i1}, x_{i2}, \dots, x_{in})^T$. The velocity (position change) of this particle, can be represented by another n -dimensional vector $v_i = (v_{i1}, v_{i2}, \dots, v_{in})^T$. The best previously visited position of the i^{th} particle is denoted as $p_i = (p_{i1}, p_{i2}, \dots, p_{in})^T$. Defining g as the index of the best particle in the swarm (i.e., the g -th particle is the best), and letting the superscripts denote the iteration number, the swarm is manipulated according to the following two equations [34]:

$$\begin{aligned} v_{id}^{k+1} &= wv_{id}^k + cr_1^k(p_{id}^k - x_{id}^k) + cr_2^k(p_{gd}^k - x_{id}^k) \\ x_{id}^{k+1} &= x_{id}^k + v_{id}^{k+1} \end{aligned} \tag{1}$$

where $d = 1, \dots, n$ denotes the d -dimensional vector, $i = 1, 2, \dots, N$ denotes i -particle, N is the size of the swarm, w is the “inertia weight”, c is a positive constant, called “acceleration constant”, and r_1, r_2 are random numbers, uniformly distributed in $[0, 1]$, and $k = 1, 2, \dots$ determines the iteration number.

To escape from local optimizations, “stretching” technique [31] can be used. The “stretching” on a objective functions $F(x, y)$ is defined by:

$$\begin{aligned} G(x, y^*) &= F(x, y) + \gamma_1 \|x - x^*\| (\text{sign}(F(x, y^*) - F(x^*, y^*)) + 1) \\ H(x, y^*) &= G(x, y^*) + \gamma_2 \frac{\text{sign}(F(x, y^*) - F(x^*, y^*)) + 1}{\tanh(\mu(G(x, y^*) - G(x^*, y^*)))} \end{aligned} \quad (2)$$

where γ_1, γ_2 , and μ are arbitrary chosen positive constant, and $\text{sign}(\cdot)$ defines the well known triple valued sign function.

$$\text{sign}(x) = \begin{cases} 1, & \text{if } x < 0 \\ 0, & \text{if } x = 0 \\ -1, & \text{if } x > 0 \end{cases}$$

3 A PSO Based Algorithm for FBOSF Problems

In this section, based on the ranking relationship defined by Definition 1, we present a FBOSF model, for which, a PSO based algorithm is developed.

3.1 A FBOSF Model

We use the ranking method defined by Definition 1 in the FBOSF model below because it is quite flexible to compare two fuzzy sets. When applying PSO technique, we need to choose current best candidate solutions from a swarm. In such a situation, it is possible that objective values under two solutions can not be compared. For example, there are two fuzzy objective values, $\tilde{a}$ and $\tilde{b}$ under two solutions, if some of the left cut set of $\tilde{a}$ are less than those of $\tilde{b}$, while some right cut set of $\tilde{a}$ are larger than those of $\tilde{b}$, there is not a ranking relation between $\tilde{a}$ and $\tilde{b}$. By the ranking method defined by Definition 1, we can adjust the satisfactory degree α to a point where all incomparable parts are discarded. Thus, we can use many more candidate solutions and extend the searching space. As a result, the risk of local optimality can be decreased to some extent.

Definition 2. For $x \in X \subset \mathbb{R}^n$, $y_i \in Y_i \subset \mathbb{R}^{m_i}$, $Y = (Y_1, Y_2, \dots, Y_K)^T$, $F, f : X \times Y_1 \times \dots \times Y_K \rightarrow F(\mathbb{R})$, and $i = 1, 2, \dots, K$, a FBOSF problem, in which $K(\geq 2)$ followers are involved, is defined as:

$$\begin{aligned} \min_{x \in X} F(x, y_1, y_2, \dots, y_K) &= \tilde{c}x + \sum_{i=1}^K \tilde{d}_i y_i \\ \text{subject to } \tilde{A}x + \sum_{i=1}^K \tilde{B}_i y_i &\preceq_{\alpha} \tilde{b} \end{aligned} \quad (3)$$

where $y_i (i = 1, 2, \dots, K)$, for each value of x , is the solution of the lower level problem:

$$\begin{aligned} \min_{y_i \in Y_i} f(x, y_1, y_2, \dots, y_K) &= \tilde{a}x + \sum_{i=1}^k \tilde{e}_i y_i \\ \text{subject to } \tilde{D}_i x + \tilde{C}_i y_i &\preceq_{\alpha} \tilde{g}_i \\ i &= 1, \dots, k \end{aligned} \quad (4)$$

where $\tilde{c}, \tilde{a} \in F(\mathbb{R}^n)$, $\tilde{d}_i, \tilde{e}_i \in F(m^i)$, $\tilde{b} \in F(\mathbb{R}^p)$, $\tilde{A} \in F(\mathbb{R}^{p \times n})$, $\tilde{B}_i \in F(\mathbb{R}^{p \times m_i})$, $\tilde{D}_i \in F(\mathbb{R}^{q \times n})$, $\tilde{C}_i \in F(\mathbb{R}^{q \times m_i})$, $\tilde{g}_i \in F(\mathbb{R}^q)$, $i = 1, 2, \dots, K$.

3.2 A PSO Based Algorithm

In this section, we apply PSO technique on FBOSF problems and develop an algorithm accordingly.

Figure 1 shows the outline of this PSO based algorithm. It first samples the leader-controlled variables to get some candidate choices for a leader. Then, we use PSO method together with Stretching technology [31] to get followers' response for every leader's choice. Thus a pool of candidate solutions for both the leader and the follower is formed. By pushing every solution pair moving towards current best ones, the whole solution pool is updated. Once a solution is reached for the leader, we use Stretching technology [31] to escape local optimization. We repeat this procedure by a pre-defined count and reach a final solution.

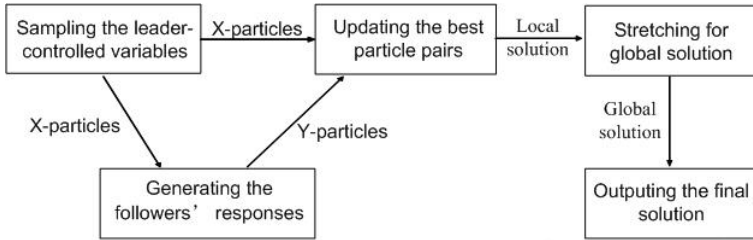


Fig. 1. The outline of the PSO based algorithm

The detailed algorithm is specified in Algorithm 1. The notations used in subsequent paragraphs are explained in Table 1.

First we initiate a swarm comprised by the leader-controlled variables (X-particles). For each particle (x_i) in the swarm, we fix x_i and pass it to the followers as a constant. Then the optimal response from the followers can be generated by solving the following single level optimization problem:

$$\begin{aligned} \min_{y_i \in Y_i} f(x, y_1, y_2, \dots, y_K) &= \tilde{a}x + \sum_{i=1}^k \tilde{e}_i y_i \\ \text{subject to } \tilde{D}_i x + \tilde{C}_i y_i &\preceq_{\alpha} \tilde{d}_i \\ i &= 1, 2, \dots, K \end{aligned} \quad (5)$$

Input: Parameters of Problem defined by Equation (3) and Equation (4)

Output: (x_g, y_g)

Sampling:

$x_i = (x_{i1}, x_{i2}, \dots, x_{in})^T$;

$v_i = (v_{i1}, v_{i2}, \dots, v_{in})^T, i = 1, \dots, N$;

Generating the responses from the follower:

foreach x_i **do**

$k_f = 1$;

$p_{ij} = (p_{ij1}, p_{ij2}, \dots, p_{ijm})^T = (0, 0, \dots, 0)^T$;

Sampling:

$y_{ij} = (y_{ij1}, y_{ij2}, \dots, y_{ijm})^T$;

$v_{ij} = (v_{ij1}, v_{ij2}, \dots, v_{ijm})^T, j = 1, \dots, M$;

$CS_i = \text{false}$;

if $\tilde{A}_1x + \tilde{B}_1y \preceq \tilde{b}_1$ **and** $\tilde{A}_2x_i + \tilde{B}_2y \preceq \tilde{b}_2$ **then**

$CS_i = \text{true}$;

end

if $(p_{ij} = (p_{ij1}, p_{ij2}, \dots, p_{ijm})^T = (0, 0, \dots, 0)^T)$ **or** $(f(x_i, y_{ij}) \leq f(x_i, p_{ij}))$ **then**

$p_{ij} = (p_{ij1}, p_{ij2}, \dots, p_{ijm})^T = (y_{ij1}, y_{ij2}, \dots, y_{ijm})^T$;

end

 Stretching for global solution for follower by (2);

 Searching the best response y_i from $p_{ij}, j = 1, 2, \dots, M$;

 Updating velocities and positions using Equation (6);

$k_f = k_f + 1$;

if $k_f \geq \text{Max}K$ **then**

 Goto 2;

else

 Goto 1;

end

end

2 if $CS_i = \text{true}$ **then**

if $(p_i = (p_{i1}, \dots, p_{in})^T = (0, \dots, 0)^T)$ **or** $(F(x_i, y_i) \leq F(p_i, y_{pi}))$ **then**

$p_i = (p_{i1}, p_{i2}, \dots, p_{in})^T = (x_{i1}, x_{i2}, \dots, x_{in})^T$;

$y_{pi} = (y_{pi1}, y_{pi2}, \dots, y_{pim})^T = (y_{i1}, y_{i2}, \dots, y_{im})^T$;

end

end

Stretching for global solution for leader by (2);

Searching (x_g, y_g) from p_i , and $y_{pi}, i = 1, \dots, N$;

Updating x_i and v_i using Equation (7);

$k_l = k_l + 1$;

if $k_l \geq \text{Max}K$ **then**

 stop

else

 Goto 2;

end

Algorithm 1. A PSO based algorithm for FBOSF problems

Table 1. The explanation of some notations

N	the number of candidate solutions (particles) by the leader within its swarm;
M	the number of candidate solutions (particles) by the followers within its swarm;
m	$= \sum_{i=1}^K m_i$, the total number of decision variables from followers;
x_i	$= (x_{i1}, x_{i2}, \dots, x_{in})^T$, $i = 1, \dots, N$, the i^{th} candidate solution for the leader;
v_i	$= (v_{i1}, v_{i2}, \dots, v_{in})^T$, $i = 1, \dots, N$, the velocity of x_i ;
y_i	$= (y_{i1}, y_{i2}, \dots, y_{im})^T$, the followers' choice for each x_i from the leader;
y_{ij}	$= (y_{ij1}, y_{ij2}, \dots, y_{ijm})^T$, $j = 1, \dots, M$, the j^{th} candidate solution by the followers for the choice x_i from the leader;
v_{ij}	$= (v_{ij1}, \dots, v_{ijm})^T$, $j = 1, \dots, M$, the velocity of y_{ij} ;
p_i	$= (p_{i1}, p_{i2}, \dots, p_{in})^T$, the best previously visited position of x_i ;
p_{ij}	$= (p_{ij1}, p_{ij2}, \dots, p_{ijm})^T$, the best previously visited position of y_{ij} ;
y_{pi}	$= (y_{pi1}, y_{pi2}, \dots, y_{pim})^T$, the response from the followers for the choice p_i from the leader;
CS	$= (CS_1, CS_2, \dots, CS_n)$, the recording vector to record if x_i is within constraint area;
g	the index of the best particle for the leader in the swarm;
k_l	current iteration number for the upper-level problem;
k_f	current iteration number for the lower-level problem;
$MaxK_l$	the predefined max iteration number for k_l ;
$MaxK_f$	the predefined max iteration number for k_f .

To solve problem (5), we also need to generate a population ($Y\_particles$), each of which has a velocity. Both the ($Y\_particles$) and the corresponding velocities are random number distributed among a pre-defined range. The followers thus have many candidate solutions of $(x_i, y_{ij}, i = 1, 2, \dots, K, j = 1, 2, \dots, K)$. From every particle pair (x_i, y_{ij}) , a bunch of the followers' objective values can be generated, which are inevitably fuzzy numbers. These fuzzy objective values will be evaluated by comparing any two of them using Definition 1. By this ranking method, we can select the previously visited best positions for all $y\_particles$ and the best one among $y\_particles$. Then stretching technology will be used to erase local solutions. Having current best positions, we adjust the velocities which are redirected towards these best positions. Then every $y\_particle$ will be moved by its corresponding velocity. Specifically, we use the following equations to update the position (y_{ij}) and velocity (v_{ij}) for each in $Y\_particles$:

$$\begin{aligned}
 v_{ij}^{k_f+1} &= wv_{ij}^{k_f} + cr_1^{k_f}(p_{ij} - y_{ij}^{k_f}) + cr_2^{k_f}(y_i^{k_f} - y_{ij}^{k_f}) \\
 y_{ij}^{k_f+1} &= y_{ij}^{k_f} + v_{ij}^{k_f+1}
 \end{aligned} \tag{6}$$

Here, k_f is to record current loop. Once k_f is larger than some predefined value, y_i will be sent to the leader as the followers' response for x_i .

Having obtained the responses of $y_i, i = 1, 2, \dots, K$, from the followers, the leader's objective values for each x_i can be calculated. We then use the fuzzy

ranking method defined by Definition 1 again to compare these objective values and select the best position for each x_i and the best one among them, which can make the most optimal objective values. After using stretching technology on current found best ones, the PSO technique is applied for every particle pair (x_i, y_i) to update the position (x_i) and velocity (v_i) of every leader's particle:

$$\begin{aligned} v_i^{k_l+1} &= wv_i^{k_l} + cr_1^{k_l}(p_i - x_i^{k_l}) + cr_2^{k_l}(x_g^{k_l} - x_i^{k_l}) \\ x_i^{k_l+1} &= x_i^{k_l} + v_i^{k_l+1} \end{aligned} \quad (7)$$

Once the iteration times (k_l) is large enough, current best particle pair (x_g, y_g) will be outputted as the final solution.

4 Conclusions

In this study, a satisfactory degree based FBOSF model and a PSO based algorithm have been developed to support the bilevel decision making with fuzzy parameters and multiple followers who share the same objective but have individual decision variables and constraints. Future research will consider the situation when more complicated relationship exists among followers, such as the cooperative or semi-cooperative decision variables and incooperative objectives.

Acknowledgments

The work presented in this paper was supported by Australian Research Council (ARC) under discovery grant DP0557154.

References

1. Von Stackelberg, H.: Theory of the Market Economy. Oxford University Press, New York (1952)
2. Bard, J.F.: Practical bilevel optimization: algorithms and applications. Kluwer Academic Publishers, Boston (1998)
3. Yu, H., Dang, C., Wang, S.: Game theoretical analysis of buy-it-now price auctions. International Journal of Information Technology and Decision Making 5(3), 557–581 (2006)
4. Hobbs, B.F., Metzler, B., Pang, J.S.: Strategic gaming analysis for electric power system: an mpec approach. IEEE Transactions on Power System 15, 637–645 (2000)
5. Zhang, G., Lu, J.: Model and approach of fuzzy bilevel decision making for logistics planning problem. Journal of Enterprise Information Management 20(2), 178–197 (2007)
6. Amat, J., McCarl, B.: A representation and economic interpretation of a two-level programming problem. Journal of the Operational Research Society 32, 783–792 (1981)
7. Feng, C., Wen, C.: Bi-level and multi-objective model to control traffic flow into the disaster area post earthquake. Journal of the Eastern Asia Society for Transportation Studies 6, 4253–4268 (2005)

8. Gao, Y., Zhang, G., Lu, J., Gao, S.: A bilevel model for railway train set organizing optimization. In: 2007 International Conference on Intelligent Systems and Knowledge Engineering (ISKE 2007), pp. 777–782. Atlantis Press (2007)
9. Bard, J.F., Moore, J.T.: A branch and bound algorithm for the bilevel programming problem. *SIAM Journal on Scientific and Statistical Computing* 11, 281–292 (1990)
10. Shi, C., Lu, J., Zhang, G.: An extended kth-best approach for linear bilevel programming. *Applied Mathematics and Computation* 164(3), 843–855 (2005)
11. Lu, J., Shi, C., Zhang, G., Ruan, D.: An extended branch and bound algorithm for bilevel multi-follower decision making in a referential-uncooperative situation. *International Journal of Information Technology and Decision Making* 6(2), 371–388 (2007)
12. Shi, C., Lu, J., Zhang, G.: An extended kuhn-tucker approach for linear bilevel programming. *Applied Mathematics and Computation* 162, 51–63 (2005)
13. Shi, C., Lu, J., Zhang, G., Zhou, H.: An extended branch and bound algorithm for linear bilevel programming. *Applied Mathematics and Computation* 180(2), 529–537 (2006)
14. Li, X., Tian, P., Min, X.: A hierarchical particle swarm optimization for solving bilevel programming problems. In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L.A., Żurada, J.M. (eds.) *ICAISC 2006*. LNCS, vol. 4029, pp. 1169–1178. Springer, Heidelberg (2006)
15. White, D., Anandalingam, G.: A penalty function approach for solving bi-level linear programs. *Journal of Global Optimization* (3), 397–419 (1993)
16. Gao, Y., Zhang, G., Lu, J.: A particle swarm optimization based algorithm for fuzzy bilevel decision making. In: *IEEE International Conference on Fuzzy Systems*, pp. 1452–1457 (2008)
17. Lu, J., Shi, C., Zhang, G.: On bilevel multi-follower decision making: General framework and solutions. *Information Sciences* 176(11), 1607–1627 (2006)
18. Kitayama, S., Yasuda, K.: A method for mixed integer programming problems by particle swarm optimization. *Electrical Engineering in Japan* 157(2), 40–49 (2006)
19. Sakawa, M., Nishizaki, I., Uemura, Y.: Interactive fuzzy programming for multilevel linear programming problems with fuzzy parameters. *Fuzzy Sets and Systems* 109, 3–19 (2000)
20. Zhang, G., Lu, J., Dillon, T.: An approximation kuhn-tucker approach for fuzzy linear bilevel decision making problems. In: Jain, L., Wren, G. (eds.) *Intelligent Decision Making*. Springer, Heidelberg (2007)
21. Zhang, G., Lu, J., Dillon, T.: Kth-best algorithm for fuzzy bilevel programming. In: *Proceedings of International Conference on Intelligent Systems and Knowledge Engineering*, Shanghai (2006)
22. Zhang, G., Lu, J., Dillon, T.: An approximation branch-and-bound algorithm for fuzzy bilevel decision making problems. In: *Proceedings of The 1st International Symposium Advances in Artificial Intelligence and Applications*, Poland (2006)
23. Gao, Y., Zhang, G., Lu, J., Zeng, X.: A λ -cut approximate approach to supporting fuzzy goal based bilevel decision making in risk management. In: *The First International Conference on Risk Analysis and Crisis Response*, pp. 132–137. Atlantis Press (2007)
24. Lu, J., Shi, C., Zhang, G., Dillon, T.: Model and extended kuhn-tucker approach for bilevel multi-follower decision making in a referential-uncooperative situation. *International Journal of Global Optimization* 38(4), 597–608 (2007)
25. Mathieu, R., Pittard, L., Anandalingam, G.: Genetic algorithm based approach to bi-level linear programming. *Recherche Operationnelle* 28(1), 1–21 (1994)

26. Pei, Z., Tian, S., Huang, H.: A novel method for solving nonlinear bilevel programming based on hybrid particle swarm optimization. In: The 8th International Conference on Signal Processing, vol. 3 (2006)
27. Zhao, Z., Gu, X.: Particle swarm optimization based algorithm for bilevel programming problems. In: 2006 6th International Conference on Intelligent Systems Design and Applications (2006)
28. Oduguwa, V., Roy, R.: Bi-level optimisation using genetic algorithm. In: Proceedings of the 2002 IEEE International Conference on Artificial Intelligence Systems, ICAIS 2002 (2002)
29. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: Proc IEEE International Conference on Neural Networks, pp. 1942–1948 (1995)
30. Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. In: Proceedings Sixth International Symposium on Micro Machine and Human Science (1995)
31. Parsopoulos, K.E., Vrahatis, M.N.: Recent approaches to global optimization problems through particle swarm optimization. *Natural Computing* 1, 235–306 (2002)
32. Zhang, J.-R., Zhang, J., Lok, T.-M., Lyu, M.R.: A hybrid particle swarm optimization-back-propagation algorithm for feedforward neural network training. *Applied Mathematics and Computation* 185(2), 1026–1037 (2007)
33. Ho, S.L., Yang, S., Ni, G., Lo, E.W.C., Wong, H.C.: A particle swarm optimization-based method for multiobjective design optimizations. *IEEE Transactions On Magnetics* 41(5), 557–581 (2005)
34. Eberhart, R.C., Simpson, P., Dobbins, R.: *Computational Intelligence PC Tools*. Academic Press, London (1996)

Reference Point-Based Particle Swarm Optimization Using a Steady-State Approach

Richard Allmendinger¹, Xiaodong Li², and Jürgen Branke¹

¹ University of Karlsruhe, Institute AIFB, Karlsruhe, Germany

² RMIT University, School of Computer Science and Information Technology,
Melbourne, Australia

Abstract. Conventional multi-objective Particle Swarm Optimization (PSO) algorithms aim to find a representative set of Pareto-optimal solutions from which the user may choose preferred solutions. For this purpose, most multi-objective PSO algorithms employ computationally expensive comparison procedures such as non-dominated sorting. We propose a PSO algorithm, Reference point-based PSO using a Steady-State approach (RPSO-SS), that finds a preferred set of solutions near user-provided reference points, instead of the entire set of Pareto-optimal solutions. RPSO-SS uses simple replacement strategies within a steady-state environment. The efficacy of RPSO-SS in finding desired regions of solutions is illustrated using some well-known two and three-objective test problems.

1 Introduction

The use of Particle Swarm Optimization (PSO) in multi-objective problem solving has become more popular in recent years [1,2,3]. The aim of such multi-objective optimization techniques is to obtain a set of trade-off solutions with respect to multiple conflicting objectives. However, using Evolutionary Multi-Objective Optimization (EMOO) approaches to find the entire set of optimal solutions on problems that have a large number of objectives may lead to a much higher computational cost [4,5]. This is especially true for the widely used NSGA-II [6], which uses a non-dominated sorting procedure that has a complexity of $O(N^2)$ (N is the population size). Since a decision maker (DM) is perhaps only interested in few (rather than all) Pareto-optimal solutions, much computational cost can be saved by focusing search on the preferred region of the Pareto-front. For this purpose, preference-based approaches such as the goal attainment method [7], the ε -constraint method [8] or the reference point-based approach [9] are drawn upon. The disadvantage of using these classical problem solving methods is that they are designed to find in each run only one single optimal solution instead of several. Given that the DM is interested in more than one solution to choose from, the classical approaches must be enhanced.

A solution to this problem would be to use population-based approaches such as multi-objective Evolutionary Algorithms (EAs) and PSO algorithms such as the Non-dominated Sorting PSO (NSPSO) [2]. In NSPSO, each particle needs

to choose a leader, which is done by finding the first non-dominated front, using the non-dominated sorting procedure, and choosing a particle randomly from the least crowded area of the first non-dominated front. This approach can be computationally expensive, especially for a large number of objectives, which is not necessarily the result of a high number of fitness evaluations but the result of the computational overhead added by other operations required by the optimization algorithm, such as the non-dominated sorting procedure or niching calculations. In this study, we use a steady-state approach that employs a simple replacement strategy, which avoids most of the computational overload caused by operations other than fitness evaluations. This method has been shown to be beneficial for EAs [10]. Although many multi-objective EAs exist in the literature, very few make use of preference-based search [11]. We propose a multi-objective PSO algorithm that employs a preference-based method in order to find a set of preferred Pareto-optimal solutions near user-provided reference points. Our approach, which we call Reference point based PSO using a Steady-State approach (RPSO-SS), achieves this aim in an effective and simple manner.

In Section 2 we introduce general background, including preference-based approaches, PSO and the steady-state approach. Section 3 describes the proposed algorithm, followed by numerical results and analysis in Section 4. Section 5 provides the concluding remarks.

2 Background

In this section we describe the main components used for the proposed algorithm.

2.1 Particle Swarm Optimization

PSO was developed by Kennedy and Eberhart [12], inspired by social behavior of bird flocking. PSO algorithms are simple and effective population-based stochastic techniques, which have their origin in the single objective optimization, but recently gained more popularity in the field of MOO. While several variants of multi-objective PSO have been developed [2,3,13,14], the basic algorithm is the same. A population of particles is initialized in an n -dimensional search space in which each particle $\mathbf{x}_i = (x_{i,1}, \dots, x_{i,n})$ represents a (possible) solution. At any time, each particle is aware of its current position, its own personal best position, its current velocity and the single global (or local) best position. The global (or local) best position is represented as $\mathbf{p}_g = (p_{g,1}, \dots, p_{g,n})$, and symbolizes either the best position of all particles in the population (global best) or the best position within a local neighborhood (local best) of the particle. The personal best position represents the best position found by a particle so far and is denoted as $\mathbf{p}_i = (p_{i,1}, \dots, p_{i,n})$. The velocity $\mathbf{v}_i = (v_{i,1}, \dots, v_{i,n})$ gives the position change of a particle. Equation (1) is used to calculate the new velocity for each particle. To update the new position of each particle, Equation (2) is used:

$$v_{i,j} \leftarrow \chi \cdot (v_{i,j} + r_1 \cdot (p_{i,j} - x_{i,j}) + r_2 \cdot (p_{g,j} - x_{i,j})) \quad (1)$$

$$x_{i,j} \leftarrow x_{i,j} + v_{i,j} \quad (2)$$

where $j = 1, \dots, n$ and $i = 1, \dots, N$, with N as population size. r_1 and r_2 are two random numbers, chosen from the interval $[0, \frac{\varphi}{2}]$. φ is a positive constant and equal to 4.1 [1]. χ is a constriction factor; used to control and restrict the velocity's magnitude. This, suggested by Clerc and Kennedy, PSO variant is called *Constriction Type 1''* and is often used with $\chi = 0.7298$, calculated according to $\chi = \frac{2}{2 - \varphi - \sqrt{\varphi^2 - 4 \cdot \varphi}}$ [1]. To reduce the likelihood of particles leaving the search space, a particle's velocity can be constricted by setting the maximum velocity V_{max} to the upper and lower bounds of the decision variable ranges.

2.2 Reference Point-Based Approach

Reference point-based approaches belong to the family of preference-based approaches, which consider user information in order to concentrate the search on a preferred region of the search space [8]. One of the first steps of using higher-level information to navigate the search was proposed by Kuhn and Tucker [15]. Wierzbicki was the first who used user information in the form of reference points [9]. His approach aimed to find optimal solutions near a user-provided reference point $\bar{z} = (\bar{z}_1, \dots, \bar{z}_m)$ by solving an achievement scalarizing function $s(f(\mathbf{x}))$:

$$\text{minimize } s(f(\mathbf{x})) = \max_{i=1, \dots, m} [w_i \cdot (f_i(\mathbf{x}) - \bar{z}_i)] \quad (3)$$

where a user-specified positive weighting vector $\mathbf{w} = (w_1, \dots, w_m)$ is used to scalarize the individual objectives f_i , $i = 1, \dots, m$. The drawback of these proposals is that they form a single optimization problem, which matches only the DM's specific preferences. This leads to a minimization problem that has to be repeated with a new reference point and/or weighting vector if the user is dissatisfied with the obtained solution. This issue can be easily overcome by using the notion of dominance as usually used in an EMOO algorithm.

2.3 Steady-State Approach

In a steady-state EA an offspring is generated one at a time. A replacement strategy is often employed to compare the offspring with its parents. The offspring only replaces a weaker parent. Note that this procedure results in a population size that is constant during the entire run of the algorithm. There is no notion of generation.

Mumford introduced a Simple EA for Multi-objective Optimization (SEAMO) that uses the steady-state approach. SEAMO was at first designed for the NP-hard, 0-1 multiple knapsack problem, but modified for continuous functions [10,16]. SEAMO uses a steady-state population and applies a simple elitist replacement strategy that relies on comparisons between parents and their generated offspring. More precisely, an offspring for each population member is sequentially produced by applying crossover and mutation with a randomly selected second parent. If the offspring dominates one of the parents, then the

offspring replaces the parent. In order to encourage diversity in the population, *duplicate* individuals are eliminated in SEAMO. First, the offspring is compared with every individual in the population. If the offspring is a *duplicate* to another individual in the population then it does not replace any other individual. Two individuals, $\mathbf{x}_1$ and $\mathbf{x}_2$, are deemed to be *duplicates* if $x_{1,j}-\epsilon \leq x_{2,j} \leq x_{1,j}+\epsilon$ applies to all objective function values $x_{1,j}$ and $x_{2,j}$ of $\mathbf{x}_1$ and $\mathbf{x}_2$, where ϵ is an error term.

3 RPSO-SS

The algorithm proposed here is a multi-objective PSO variant that is able to locate Pareto-optimal solutions closest to user-provided reference points. This is achieved without employing the non-dominated sorting procedure, instead we use simple replacement strategies in a steady-state environment. This section presents the basic Reference point based PSO using a Steady-State approach (RPSO-SS) (see Section 3.1) and another variant of RPSO-SS, which uses an extended replacement strategy and a sampling method to choose a leader $\mathbf{p}_g$ (see Section 3.2). We first outline the proposed steady-state approach in PSO, which slightly differs from the steady-state approach for EAs. In RPSO-SS, instead of using two parents, the new position of the current particle $\mathbf{x}_i$ is generated based on a randomly selected $\mathbf{p}_g$; we select $\mathbf{p}_g$ randomly in order to maintain diversity. The new position is analogous to an offspring in conventional EAs. The replacement strategy applied to the three particles ($\mathbf{x}_i$, $\mathbf{p}_g$ and the offspring) is based on pairwise comparisons and will keep only two particles in the population so that the population size remains constant. This replacement strategy will provide the necessary selection pressure to drive the population moving towards the Pareto-optimal front. The same process is repeated for the entire population.

3.1 Basic RPSO-SS

Initialization. The DM is asked to provide his or her preferences. This includes one or more reference points and the desired spread of solutions, specified by the error term ϵ . The swarm is initialized with a population size that is constant over the entire run of the algorithm. The velocity $\mathbf{v}_i$ is set randomly to be within the variable ranges. The population is divided into equal sized clusters, each focused on one reference point. The following main loop is applied for each cluster separately.

Main Loop. Do the following for each particle $\mathbf{x}_i$ in each cluster until a stopping criterion is met:

1. Choose a particle randomly from the current cluster as the $\mathbf{p}_g$.
2. Produce an offspring based on $\mathbf{x}_i$, (the personal best position) $\mathbf{p}_i$ and $\mathbf{p}_g$ by applying Equations (1) and (2).
3. (a) If the offspring dominates $\mathbf{x}_i$, then it replaces $\mathbf{x}_i$,
 (b) else if the offspring dominates $\mathbf{p}_g$, then it replaces $\mathbf{p}_g$ (see Fig. 1),

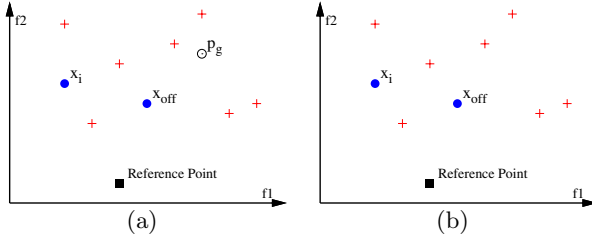


Fig. 1. An example of main loop 3.(b): a) An offspring x_{off} is first generated. b) Since x_{off} dominates particle p_g (and not x_i), x_{off} replaces particle p_g . Note that the population size remains constant during the entire procedure.

- (c) else if the offspring is non-dominated to x_i and to p_g , then the two particles closest to the reference point are kept, and the distant particle is deleted.

Our experience shows that the steady-state approach works well on locating the entire front without using non-dominated sorting. However, when reference points are considered, the population tends to converge to a single non-dominated solution that is closest to the reference point. To prevent this, an error term ϵ is introduced to control the spread of the solutions. This can be done by eliminating duplicate particles using the same procedure as used in SEAMO (see Section 2.3). Using a larger error term will lead to a more scattered distribution of particles, whereas a smaller value will result in a more concentrated search.

Moreover, to further involve the user in the search process we ask the DM to reset a reference point that is dominated by an offspring that is kept in the population. The reference point can also be reset automatically. We apply following formula: $\bar{z}_i = \bar{z}_{i, current} - |\bar{z}_{i, original} \cdot step\_width|$, in which $i = 1, \dots, m$. $\bar{z}_i$ is the new component value, $\bar{z}_{i, current}$ is the current component value and $\bar{z}_{i, original}$ the original aspiration level of objective i of reference point $\bar{z}$; $step\_width$ specifies how far the reference point will be reset with respect to the original provided reference point $\bar{z}_{i, original}$. Note that this procedure leads to a new reference point that always dominates the old one.

3.2 An Extended Replacement Strategy with Sampling

The replacement strategy used in the basic RPSO-SS won't keep an offspring in the population that has a larger Euclidean distance than x_i and p_g to the reference point (see Section 3.1, Step 3.(c) of the main loop). Perhaps it would make more sense if such an offspring were allowed to replace another particle x_r . Replacement strategy with sampling extends the replacement strategy of the basic RPSO-SS and provides the offspring with the opportunity to replace particles other than x_i or p_g . Here, a randomly selected particle x_r will be replaced if it is either dominated by the offspring, or if it is non-dominated to the offspring and has a larger Euclidean distance than the offspring to the reference point. If

no such $\mathbf{x}_r$ can be found in the population, then the offspring won't replace any particle.

Instead of using a single randomly selected particle as a $\mathbf{p}_g$, we can sample several particles randomly. Among these particles, the one with the shortest Euclidean distance to the reference point is chosen as the $\mathbf{p}_g$. In the special case where the sample size is equal to one, the single particle is always chosen as $\mathbf{p}_g$. Obviously, a larger sample size would increase the chance of finding a better $\mathbf{p}_g$, at the cost of higher computational effort. In our study here, we investigate the performance of two different sample sizes: size = 1 and size = 5.

4 Experiments

For validation purposes, the basic RPSO-SS and the extended replacement strategy with sample size = 1 (denoted by RS-S1) and size = 5 (denoted by RS-S5) were examined on some well-known test problems. ZDT1, ZDT2, ZDT3 and ZDT4 [17] were chosen to investigate the performance on two-objective problems. For the three-objective problems, DTLZ2 and DTLZ6 [18] were chosen. For the ZDT functions, we used a population size of 200 and terminated the simulation by 600 iterations. The DTLZ functions were run for 800 iterations using a population size of 400. For all experiments the reference points can be reset with a *step_width* = 0.25 if they are in the feasible space.

To measure the performance we computed the normalized hypervolume (NH) of the last iteration for RPSO-SS, RS-S1 and RS-S5. NH is obtained by evaluating the hypervolume [19] of a normalized set of non-dominated solutions. In order to normalize a solution set an *ideal* and a *nadir point* are required. The nadir point and the ideal point are defined by the maximal and minimal objective function values, respectively, of the union of all non-dominated sets obtained across all runs of all comparing variants (RPSO-SS, RS-S1 and RS-S5). Our approaches were run for 50 separate runs. In order to provide fair comparisons of the NH value, the same nadir and ideal point are used across the solution sets obtained for a particular problem. The aim is to maximize the NH. The advantage of the NH metric is that the diversity and the closeness of the found solutions can be expressed by just one numerical value, and that a known Pareto-optimal set is not required.

4.1 Results and Discussion

The procedure for obtaining preferred Pareto-optimal solutions closest to the reference points has a complexity of $O(N)$. Found solutions using this procedure for ZDT1 are shown in Fig. 2 (indicated with $\epsilon = 0.0x_i$) where two reference points ((0.2, 0.4) and (0.6, 0.5)) are chosen. If we adopt the procedure for checking duplicates using the ϵ -parameter (see Section 3.1) we will be able to control the spread of the solutions. However, this procedure has a complexity of $O(N^2)$. Fig. 2 also shows the spreads of solutions using three different ϵ values of $0.005x_i$, $0.01x_i$ and $0.03x_i$. Solutions with $\epsilon = 0.0x_i$ are shown on the true Pareto-optimal

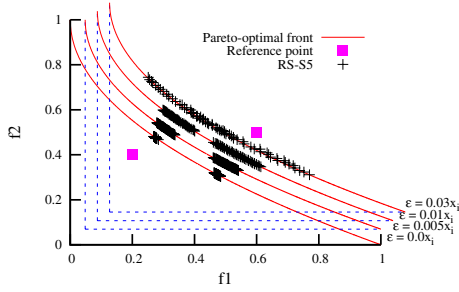


Fig. 2. Effect of different ϵ values on ZDT1

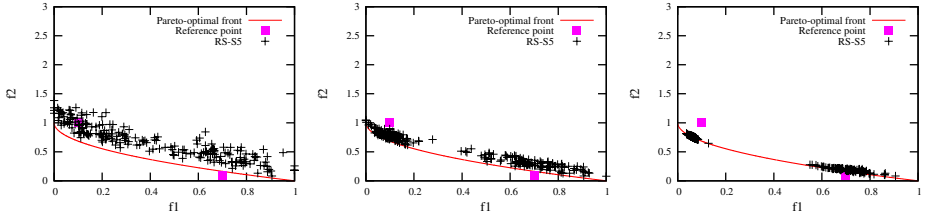


Fig. 3. Snapshots of RS-S5 on ZDT1 for iteration 10, 25 and 350 (from left to right)

front. Solutions with other ϵ values are shown with an offset to the true Pareto-optimal front. For all following experiments we set the ϵ at $0.005x_i$ for every component of the objective variable $\mathbf{x}$. Fig. 3 shows some snapshots of a single run of RS-S5 on ZDT1. It can be seen that after just a few iterations two clusters have been formed around the provided reference points ((0.1, 1.0) and (0.7, 0.1)).

Next we consider ZDT2, which has a smooth concave front. As shown in Fig. 4a, one reference point (0.4, 0.7) is chosen. It can be seen that the Pareto-optimal fronts of all compared variants have a similar distribution of non-dominated solutions. The slightly larger spread of RS-S5 was observed for all investigated problems.

On ZDT3, which has a discontinuous Pareto-front, none of the strategies had any difficulties finding efficient preferred solutions. As shown in Fig. 4b, three reference points are chosen ((0.2, 0.3), (0.5, -0.1) and (0.9, -0.6)). It can also be seen that for the reference point, which is lying between two disconnected sections, efficient solutions on both sections have been found.

For the highly multi-modal problem ZDT4, we chose one reference point (0.9, 0.4). It seems to be a challenging problem to existing multi-objective PSO algorithms, including NSPSO, which tends to get stuck on one of the many local Pareto-fronts. None of our variants performed well on this problem. However, when sample size is increased to 20 (RS-S20), the variant was able to find the 2nd best front with most particles lying close to the reference point (see Fig. 4c). This reveals that a larger sample size significantly affects the outcome.

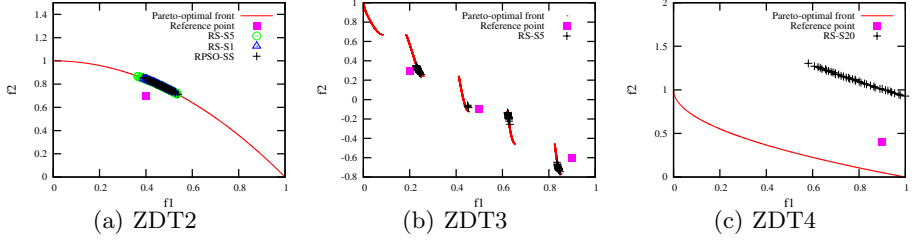


Fig. 4. Non-dominated solutions found on ZDT2, ZDT3 and ZDT4. On ZDT3 and ZDT4 only the results obtained by RS-S5 and RS-S20 are shown respectively.

Test problem DTLZ2 and DTLZ6 have a non-convex frontier and a Pareto-optimal front that is a curve, respectively. We use two reference points ((0.2, 0.4, 0.7) and (0.8, 0.8, 0.6)) for test function DTLZ2 (see Fig. 5a), and three ((0.1, 0.1, 0.9), (0.55, 0.55, 0.8) and (0.6, 0.6, 0.1)) for DTLZ6 (see Fig. 5b). Two and three well-separated sets of solutions without outliers are discovered for DTLZ2 and DTLZ6, respectively. From Fig. 5b it is apparent what was already indicated in Fig. 4a for test problem ZDT2, there is a larger spread of solutions found by RS-S5. The distribution of solutions found by RPSO-SS and RS-S1 are very similar.

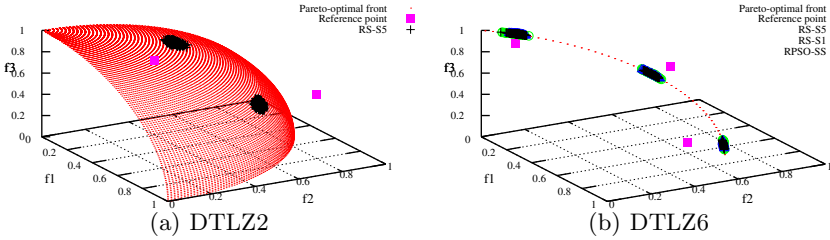


Fig. 5. a) Non-dominated solutions found by RS-S5 on DTLZ2. b) Non-dominated solutions found by RPSO-SS, RS-S1 and RS-S5 on DTLZ6.

To confirm the visual results we calculated the NH and reported the mean and the standard deviation among 50 runs for all test problems, except ZDT4. In addition, we calculated the NH for the ZDT and DTLZ test problems using a population size of 100 and 200, respectively. Note that the same ideal and nadir point are used across the solution sets obtained for both of the population sizes on a particular problem. From the summarized results of Table 1 it can be seen that in terms of the NH, RS-S5 has the largest NH values for all test problems, whereas RPSO-SS and RS-S1 have similar values. This conforms the spread distributions observed in Fig. 4a for test problem ZDT2 and in Fig. 5b for DTLZ6. Furthermore, although the NH values are very similar for a particular problem with respect to the population sizes, there is a strong tendency that the NH increases with the population size. We attribute the superior result of RS-S5

Table 1. Mean and variance values of the normalized hypervolume for population size of 100 and 200 on ZDT test problems, and 200 and 400 for DTLZ test problems (bold indicates the best results obtained on a problem for a particular population size)

Population Size	Problem	RPSO-SS		RS-S1		RS-S5	
		Mean	STD	Mean	STD	Mean	STD
100	ZDT1	0.2818	0.0042	0.2821	0.0128	0.3178	0.0683
	ZDT2	0.4187	0.1530	0.4035	0.1248	0.5228	0.1750
	ZDT3	0.5585	0.0063	0.5576	0.0015	0.5635	0.0174
200	ZDT1	0.3133	0.0131	0.3074	0.0036	0.3321	0.0356
	ZDT2	0.3921	0.0456	0.3843	0.0213	0.4630	0.0582
	ZDT3	0.5688	0.0176	0.5643	0.0010	0.5727	0.0140
200	DTLZ2	0.3550	0.0225	0.3593	0.0362	0.4311	0.0566
	DTLZ6	0.2701	0.0087	0.2667	0.0015	0.2901	0.0582
	DTLZ7	0.2184	0.0353	0.2098	0.0273	0.2537	0.0882
400	DTLZ2	0.3604	0.0303	0.3488	0.0217	0.4305	0.0740
	DTLZ6	0.2777	0.0130	0.2730	0.0011	0.2999	0.0688
	DTLZ7	0.2222	0.0541	0.2132	0.0344	0.2929	0.1012

to the fact that the sampling method provides a more reliable way of choosing a leader. This results in a greater selection pressure, whereas that leads to a better convergence towards the reference point.

5 Conclusions

We have proposed a multi-objective PSO algorithm that finds a set of Pareto-optimal solutions near user-provided reference points, using a steady-state approach. We have suggested two different variants, the basic RPSO-SS and a variant that uses an extended replacement strategy and a sampling method to choose a leader p_g . Both variants manage to find preferred solutions without using the non-dominated sorting procedure or niching calculations, with the variant using a sample size greater than one having the best convergence. In order to avoid duplicates in the population, but above all to control the extent of the spread of obtained solutions, a parameter ϵ is incorporated. This parameter specifies how close two population members can be located in the objective space without being deemed to be duplicates. This parameter can be used in conjunction with a method to automatically reset the reference point (or manually by the decision maker), which may further improve the performance of RPSO-SS. Our results on some well-known test problems demonstrated that the proposed approach is able to discover preferred areas of the Pareto-front. Work is in progress on improving the performance on multi-modal functions, such as ZDT4. The development of more efficient replacement and leader selection strategies is another aim that we are pursuing.

References

1. Clerc, M., Kennedy, J.: The particle swarm-explosion, stability, and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computation 6(1), 58–73 (2002)

2. Li, X.: A non-dominated sorting particle swarm optimizer for multiobjective optimization. In: *Proceedings of Genetic and Evolutionary Computation Conference 2003 (GECCO 2003)*, pp. 37–48 (2003)
3. Li, X.: Better spread and convergence: Particle swarm multiobjective optimization using the maximin fitness function. In: Deb, K., et al. (eds.) *GECCO 2004*. LNCS, vol. 3103, pp. 117–128. Springer, Heidelberg (2004)
4. Deb, K., Chaudhurid, S., Miettinen, K.: Estimating nadir objective vector quickly using evolutionary approaches. Technical Report No. 2005009, KanGAL, Kanpur India (2005)
5. Corne, D.W., Knowles, J.D.: Techniques for highly multiobjective optimisation: Some nondominated points are better than others. In: *Proceedings of Genetic and Evolutionary Computation Conference 2007 (GECCO 2007)*, pp. 773–780 (2007)
6. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6(2), 182–197 (2002)
7. Gembicki, F.W.: Vector optimization for control with performance and parameter sensitivity indices. PhD thesis, Case Western Reserve University, Ohio, USA (1974)
8. Chankong, V., Haimes, Y.Y.: *Multiobjective Decision Making Theory and Methodology*. Elsevier Science Publishing Co., Amsterdam (1983)
9. Wierzbicki, A.P.: The use of reference objectives in multiobjective optimisation. *Multiple Criteria Decision Making Theory and Applications*, 468–486 (1980)
10. Mumford-Valenzuela, C.L.: A simple approach to evolutionary multiobjective optimization. *Evolutionary Computation Based Multi-Criteria Optimization: Theoretical Advances and Applications*, 55–79 (2004)
11. Coello Coello, C.A.: Handling preferences in evolutionary multiobjective optimization: A survey. In: *Proceedings of the 2000 Congress on Evolutionary Computation (CEC 2000)*, pp. 30–37 (2000)
12. Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. In: *Proceedings of the Sixth International Symposium on Micro Machine and Human Science (MHS 1995)*, pp. 39–43 (1995)
13. Parsopoulos, K.E., Vrahatis, M.N.: Particle swarm optimization method in multiobjective problems. In: *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC 2002)*, pp. 603–607 (2002)
14. Sierra, M.R., Coello Coello, C.A.: Improving PSO-based multi-objective optimization using crowding, mutation and ϵ -dominance. In: *Proceedings of the Third Evolutionary Multi-Criterion Optimization Conference (EMO 2005)*, pp. 505–519 (2005)
15. Kuhn, H.W., Tucker, A.W.: Nonlinear programming. In: *Proceedings of Second Berkeley Symposium on Mathematical Statistics and Probability*, pp. 481–492 (1950)
16. Mumford-Valenzuela, C.L.: Simple population replacement strategies for a steady-state multi-objective evolutionary algorithm. In: *Proceedings of Genetic and Evolutionary Computation Conference 2004 (GECCO 2004)*, pp. 1389–1400 (2004)
17. Zitzler, E., Deb, K., Thiele, L.: Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation* 8(2), 173–195 (2002)
18. Deb, K., Thiele, L., Laumanns, M., Zitzler, E.: Scalable test problems for evolutionary multi-objective optimization. Technical Report No. 112, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland (2001)
19. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation* 3(4), 257–271 (1999)

Genetic Algorithm Based Methods for Identification of Health Risk Factors Aimed at Preventing Metabolic Syndrome

Topon Kumar Paul^{1,*}, Ken Ueno¹, Koichiro Iwata², Toshio Hayashi²,
and Nobuyoshi Honda²

¹ Corporate Research & Development Center, Toshiba Corporation
1, Komukai-Toshiba-cho, Saiwai-ku, Kawasaki 212-8582, Japan
{[toponkumar.paul](mailto:toponkumar.paul@toshiba.co.jp),[ken.ueno](mailto:ken.ueno@toshiba.co.jp)}@toshiba.co.jp

² Toshiba Corporation
1-1, Shibaura 1-chome, Minato-ku, Tokyo 105-8001, Japan
{[koichiro.iwata](mailto:koichiro.iwata@toshiba.co.jp),[toshio7.hayashi](mailto:toshio7.hayashi@toshiba.co.jp),[nobuyoshi.honda](mailto:nobuyoshi.honda@toshiba.co.jp)}@toshiba.co.jp

Abstract. In recent years, metabolic syndrome has emerged as a major health concern because it increases the risk of developing lifestyle diseases, such as diabetes, hypertension, and cardiovascular disease. Some of the symptoms of the metabolic syndrome are high blood pressure, decreased HDL cholesterol, and elevated triglycerides (TG). To prevent the developing of metabolic syndrome, accurate prediction of the future values of these health risk factors and identification of other factors from the health checkup and lifestyle data, which are highly related with these risk factors, are very important. In this paper, we propose a new framework, based on genetic algorithm and its variants, for identifying those important health factors and predicting the future health risk of a person with high accuracy. We show the effectiveness of the proposed system by applying it to the health checkup and lifestyle data of Toshiba Corporation.

Keywords: Feature selection, classification, unbalanced data, metabolic syndrome, fitness evaluation, RPMBGA+, AUC balanced.

1 Introduction

Metabolic syndrome is a group of health conditions that increase the risk of developing lifestyle diseases, such as diabetes, hypertension, and cardiovascular disease. High blood pressure (BP), decreased HDL cholesterol, elevated triglycerides (TG), high blood sugar levels, and obesity are some of the symptoms and features of the metabolic syndrome [1]. It has been reported that the development of the metabolic syndrome can be prevented or delayed by changing the lifestyle of a person. In this context, given the current health condition and lifestyle, very accurate prediction of the future levels of the health risk factors (BP, HDL cholesterol, TG, and so on) and the identification of other factors highly related with these risk factors are very important.

* Corresponding author.

To design a computational model for the prediction of the health risk of a person, we need a classifier and a feature subset selection method. The feature subset selection method generates a set of candidate feature subsets and the classifier classifies the data of a feature subset and returns the classification statistics, namely, the numbers of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN), which are utilized by a score calculation method that returns the goodness score of the classifier. As a feature subset selection method, genetic algorithm (GA) [2][3], genetic programming (GP) [4], and feature ranking, to name a few prominent ones, have been used [5,6,7,8,9,10]. As a classifier, genetic programming, k-nearest neighbor (kNN) classifier, naive-Bayes classifier, decision tree (C4.5) [11], neural network, and support vector machine (SVM) [12] are widely used [5,7,9,10,13,14].

There are many challenges for the tasks. First, the data are unbalanced; the number of persons having high health risk is very small compared to the number of persons having normal health condition. Hereafter, we refer the group of the persons having high health risk as the *minority class*, the group of persons having normal health condition as the *majority class*, and the levels of the target health risk factor as the *class labels*. The health checkup and lifestyle data of a person is referred to as an *instance*. We use *factor* and *feature* interchangeably to mean either a screening test name or a subject of lifestyle. Very accurate prediction of the future risk of a person using a model that is learned on unbalanced data is very difficult because during the learning of the model, the majority class biases the model toward it, and the learned model very often fails to accurately predict the class label of an instance from the minority class. In some cases, it has been found that all the instances from the majority class are correctly classified by the model but none of the instances from the minority is correctly classified. If a model is designed by focusing on the accurate prediction of the class labels of the instances from the minority class, the model will misclassify a very large number of instances from the majority class. That is, most traditional methods face a trade-off between the accurate prediction of the class label of an instance from the majority and the minority classes—when the prediction rate of one class increases, the prediction rate of the other class decreases. The second challenge is that the instances in the data are of variable length. Some persons may have medical history of 10 years of more; some persons, such as the new employees of an organization may have only the data of the current year. The third challenge is that the data contains a large number of features. Each year during the health checkup, if m screening tests are performed and q questions are asked about the lifestyle, the number of features in y checkups is $y(m + q)$, which becomes very large for the data of several years. To deal with these problems, we propose in this paper: three evaluation parameters, which are utilized to return the goodness score of a feature subset; two feature subset generation methods based on GA and probabilistic model building genetic algorithm (PMBGA) [15], which utilize the known information about the relationship of the other factors with target health risk factor; and a new method for calculation of distance between two variable-length instances in the kNN classifier.

2 Methods

To select a highly informative subset of factors, we use genetic algorithm based methods, which are population based global search heuristics based on artificial selection and evolution. There are a number of issues associated with genetic algorithm based methods; in this section, we only present the methods used to generate candidate feature subsets, evaluate a subset of features, and classify the variable length data using the kNN classifier in details.

2.1 Evaluation of a Subset of Features

In classification of data, accuracy is widely used as the evaluation measure of a classifier. However, in unbalanced data, the accuracy is heavily biased by the majority class and thus is a useless index of measure of goodness of a classifier. During classification of unbalanced data, a classifier or a feature subset should be evaluated in terms of both sensitivity and specificity. Some of the evaluation functions that take into account the sensitivity and specificity information are G-score, area under ROC curve (AUC), and F-score, which are defined below:

$$\text{G-score} = \sqrt{(\text{sensitivity} * \text{specificity})}; \quad (1)$$

$$\text{AUC} = \frac{1}{2}(\text{sensitivity} + \text{specificity}); \quad (2)$$

$$\text{F-score} = \frac{2 * \text{precision} * \text{recall}}{(\text{precision} + \text{recall})} \quad (3)$$

where $\text{sensitivity} = \text{recall} = \frac{N_{TP}}{N_{TP} + N_{FN}}$, $\text{specificity} = \frac{N_{TN}}{N_{TN} + N_{FP}}$ and $\text{precision} = \frac{N_{TP}}{N_{TP} + N_{FP}}$ with N_{TP} , N_{TN} , N_{FP} , and N_{FN} being the number of true positives, true negatives, false positives, and false negatives, respectively. However, there is always a trade-off between the sensitivity and the specificity; when we want to increase the one, the other decreases. For some applications, high sensitivity is required; for some other applications, high specificity is desired. In some cases, the misclassification costs of an instance from the majority and the minority classes are known and a classification model is desirable that minimizes the total misclassification cost. In most cases, the misclassification costs are unknown but a reasonable sensitivity is desired. Taking into account these situations, we propose three evaluation parameters (w_1, w_2, θ) , which are used to determine the appropriate score for a feature subset. In the evaluation parameters, w_1 and w_2 are the misclassification costs of an instance from the majority and the minority classes, and $\theta \in [0, 1]$ is the threshold of sensitivity of the minority class. That is, the evaluation parameters provides a vector of values of (w_1, w_2, θ) . If the misclassification costs are known, the vector of values becomes $(w_1, w_2, 0)$; if the misclassification costs are unknown, the vector becomes $(0, 0, \theta)$. Depending on the values of the evaluation parameters, a score is returned in the following way:

$$\text{Score} = \begin{cases} f(w_1, w_2, N_{TP}, N_{TN}, N_{FP}, N_{FN}) & \text{if } (w_1 > 0, w_2 > 0, \theta = 0); \\ f(N_{TP}, N_{TN}, N_{FP}, N_{FN}) & \text{if } (\theta > 0 \ \& \ \text{sensitivity} < \theta); \\ f(\lambda, N_{TP}, N_{TN}, N_{FP}, N_{FN}) & \text{if } (\theta > 0 \ \& \ \text{sensitivity} \geq \theta); \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

where $f(w_1, w_2, N_{TP}, N_{TN}, N_{FP}, N_{FN})$ is a cost function that returns the total cost of misclassifications; $f(N_{TP}, N_{TN}, N_{FP}, N_{FN})$ is a normal score function, such as (1), (2), or (3); and $f(\lambda, N_{TP}, N_{TN}, N_{FP}, N_{FN})$ is a function that scales up a normal score of $f(N_{TP}, N_{TN}, N_{FP}, N_{FN})$. Some examples of cost functions and the scaled up functions are as follows:

$$f(w_1, w_2, N_{TP}, N_{TN}, N_{FP}, N_{FN}) = \frac{1}{(w_1 + w_2)}(w_1 * N_{FN} + w_2 * N_{FP}); \quad (5)$$

$$f(\lambda, N_{TP}, N_{TN}, N_{FP}, N_{FN}) = \sqrt[\lambda]{f(N_{TP}, N_{TN}, N_{FP}, N_{FN})}; \quad (6)$$

$$f(\lambda, N_{TP}, N_{TN}, N_{FP}, N_{FN}) = \lambda * f(N_{TP}, N_{TN}, N_{FP}, N_{FN}) \quad (7)$$

where $f(N_{TP}, N_{TN}, N_{FP}, N_{FN}) \in [0, 1]$ is the normal score and $\lambda > 1$ is a scaled up parameter, say $\lambda = 2$.

2.2 Generation of Feature Subsets

In this paper, we propose two methods for generation of candidate feature subsets. The two methods create offspring in two different ways. In either of the methods, each candidate feature subset in the population is a vector of 0s and 1s. If a value in the vector is 1, the corresponding feature is selected; otherwise, the feature is not selected. For example, if a data set has 10 features, (1,0,1,0,1,0,1,0,0,1) is a candidate feature subset in which the first, the third, the fifth, the seventh, and the tenth features are selected.

Genetic Algorithm Based Method. Our proposed first method is an offspring generation method in genetic algorithm. New offspring are generated from the selected individuals by applying crossover and probabilistic sampling simultaneously. For each feature that is known to be associated with the target health risk factor with probability p_i , the corresponding bits in the new offspring are generated by sampling p_i ; other bits in the new offspring are generated through crossover. Let us give an example of this process. Suppose that there are 10 features in the data, and feature# 5 and 8 are known to be associated with the target health risk factor with probabilities 0.6 and 0.7, respectively. Two selected individuals from the population are (1,0,1,0,1,0,0,0,1,1) and (0,1,0,0,1,0,1,0,0,1), and a one-point crossover is to be applied on them to generate offspring. Suppose the crossover point is between bit position# 5 and 6. The sequences of steps for generation of two offspring is as follows:

1. Generate bits for position# 5 and 8 by sampling the corresponding probabilities. Suppose the sampled bits for the two offspring are: (—,—,—,—,1,—,—,1,—,—) (—,—,—,—,0,—,—,1,—,—).
2. Copy other bits from the selected individuals, which are to the left of the crossover point: (1,0,1,0,1,—,—,1,—,—), (0,1,0,0,0,—,—,1,—,—).
3. Swap the remaining bits: (1,0,1,0,1,0,1,1,0,1), (0,1,0,0,0,0,0,1,1,1).

RPMBGA+. Our proposed second method belongs to the group probabilistic model building genetic algorithm (PMBGA) [15], and is an extension of random probabilistic model building genetic algorithm (RPMBGA) [7]. We call this method RPMBGA⁺ [16]. RPMBGA⁺ is a global search heuristic like genetic algorithm but it maintains a vector of probabilities of the features in addition to a population of a set of candidate feature subsets and generates new solutions (feature subsets) by sampling the probability vector instead of using crossover and mutation operations of genetic algorithm. A value $P(X_i, t)$ in the vector of probabilities indicates the probability of the feature X_i being selected in a candidate feature subset at generation t . In our proposed method, the known information about the features is utilized during the initialization of the probability vector and the update of the probability vector. In RPMBGA⁺, the probability vector is initialized in the following ways:

$$P(X_i, 0) = \begin{cases} p_i & \text{if } p_i > 0; \\ 0.5 & \text{otherwise} \end{cases} \quad (8)$$

where p_i is the known information about the relationship of the feature with the target risk factor. When no information is known about the relationship of the feature with the target factor, the probability is set to 0.5, which means that the feature may or may not be selected as an important feature. Next, using the selected candidate feature subsets, the probability vector is updated in the following way:

$$P(X_i, t+1) = \begin{cases} p_i & \text{if } p_i > 0; \\ \psi(P(X_i, t), M(X_i, t)) & \text{otherwise} \end{cases} \quad (9)$$

where p_i is the known information about the relationship of the feature X_i with the target risk factor, $M(X_i, t)$ is the probability distribution of the feature X_i in the selected candidate feature subsets, and $\psi(P(X_i, t), M(X_i, t))$ is a function that returns a value between 0 and 1. Various methods have been proposed in literature for the update function $\psi(P(X_i, t), M(X_i, t))$; for example, in PBIL [17], $\psi(P(X_i, t), M(X_i, t))$ is defined in the following way:

$$\psi(P(X_i, t), M(X_i, t)) = \alpha P(X_i, t) + (1 - \alpha)M(X_i, t) \quad (10)$$

where $\alpha \in [0, 1]$ is called learning rate and fixed throughout each iteration. In RPMBGA [7], that function is defined as follows:

$$\psi(P(X_i, t), M(X_i, t)) = \alpha\beta P(X_i, t) + (1 - \alpha)(1 - \beta)M(X_i, t) \quad (11)$$

where $\alpha \in [0, 1]$ is called learning rate and fixed throughout each iteration, and $\beta \in [0, 1]$ is a random number and changes at each iteration. Due to the inclusion of an extra random parameter less than 1.0, $\psi(P(X_i, t), M(X_i, t))_{RPMBGA} < \psi(P(X_i, t), M(X_i, t))_{PBIL}$. Therefore, when the number of features in a data set is huge, RPMBGA will return a smaller size feature subset than PBIL. However, after a number of generations, the probability becomes so small that most of the individuals are empty feature subsets. To prevent this, we set the lower limit of $\psi(P(X_i, t), M(X_i, t))$ to a very small value, for example, to 0.001, and the value depends on the number of features in the data.

2.3 Dealing with Variable Length Data

Health checkup and lifestyle data are variable length time-series data but different from other time series data, such as waveforms. The straightforward method for making the length of each instance equal is to aggregate the values of a feature across various years into a value. This aggregation method depends on the data type of the feature. If a feature is numeric, a function that returns a numeric value should be used; if the feature is nominal, a function that returns a nominal value should be used. Some examples of aggregation functions for numeric features are *average*, *max*, and *min*; one example of aggregation function for nominal features is *mode* that returns the most frequent value. The aggregation method is usually applied as a preprocessing method and then the feature subset selection and classification methods are applied to the equal length data. In this paper, we propose a new method to select a set of important features with GA based methods and the kNN classifier. To this end, two important issues, the representation of an individual in GA based methods and the way of calculating distance in the kNN classifier, are discussed below.

Representation. If an organization collects the health checkup and lifestyle data in n categories, the length of the data of each person will be a multiple of n . We represent each candidate feature subset as an n -bit binary string. To determine the nearest neighbors, first the data of selected features are extracted from health checkup and lifestyle data of various years and then the distance between two instances are calculated using a sliding window method.

Calculation of Distance and Determination of Future Risk. To find the minimum distance position (closest matching) of an instance Y in a larger instance Z , we use sliding window method and take the window that has the nearest distance. Suppose the data of Y and Z are denoted by $\mathbf{X}_2^Y \mathbf{X}_1^Y$ and $\mathbf{X}_5^Z \mathbf{X}_4^Z \mathbf{X}_3^Z \mathbf{X}_2^Z \mathbf{X}_1^Z$ where $\mathbf{X}_i^P$ denotes the health checkup and lifestyle data of person P in the most recent year i . If the best matching portion of Y in Z is $\mathbf{X}_5^Z \mathbf{X}_4^Z$, the predicted future health risks of Y in the next three years will be the values of the health risk factor in $\mathbf{X}_3^Z$, $\mathbf{X}_2^Z$ and $\mathbf{X}_1^Z$.

3 Experiments and Results

3.1 Data

In co-operation with occupational health physicians in Toshiba Corporation, Japan, we analyze the anonymous health checkup and life style data of the employees in Toshiba Corporation by employing the proposed system. We are working on this analysis as a part of the health promotion project started in 2004. From the health checkup and lifestyle database, we create two different data sets by first filtering out the missing data and then filtering out the data of those employees who had the same high health risk (RED level, see below) in the latest two years. As the target health risk factors, we choose blood pressure

Table 1. Two data sets used in the experiments

Target health risk factor	#Features	#Majority instances	#Minority instances
Blood pressure (BP)	320	5128	1347
Triglycerides (TG)	132	4980	2046

and TG, which are known to be associated with metabolic syndrome, and the numeric values of these two factors are transformed into the three levels of RED, YELLOW, and GREEN using the scales defined by the occupational health physicians. In this paper, we treat the YELLOW level of the target health risk factor as RED level to transform the problem into a binary classification task. In either of the data sets of blood pressure and TG, for each employee, the value of the target health risk factor of the latest year is used as the class label, the data of the year preceding immediately the latest year are used as the current data, and the data of the remaining years are either used as history data (blood pressure data) or aggregated to make the length of each instance equal (TG data). The descriptions of the two data sets are given in Table 1.

3.2 Settings of Various Parameters

We perform various experiments on the data sets using baseline method (no learning; all features are used) and RPMBGA⁺. As a classifier, we use C4.5, SVM and the kNN classifier. For the kNN classifier, we use Euclidean distance during finding of the nearest neighbors of an instance and perform experiments with $k = 5$; for C4.5 and SVM, we use the WEKA [18] implementation with the default settings. We use 10-fold cross-validation to measure the overall performance of a classifier and to evaluate the goodness score of a feature subset. As a normal evaluation score, we use the geometric mean of the sensitivity and the specificity (1), and as a scaled-up score, we use (6) with $\lambda = 2$. We assume that the costs of misclassifications are unknown and set the sensitivity threshold θ to 0.5. The settings of other parameters are as follows: population size=100, elite size=2, selection size=50%, $\alpha=0.9$, and maximum number of generations per run=20. Since we use 10-fold cross-validation, the results presented here are of 10 runs (one run per fold).

Assuming no known information about the relationships of the features with blood pressure or TG, each value in the probability vector of RPMBGA⁺ is initialized with 0.5; therefore, approximately 50% of the features are selected in each individual in the initial population. For generation of new population, elitism is used; the best two feature subsets of the previous generation survive for the next generation, and the remaining 98 feature subsets are replaced with the newly generated ones. In each run, the algorithm terminates when both the sensitivity and the specificity are 1.0 (=100%) or the maximum number of generations has passed.

3.3 Results

The results obtained by applying RPMBGA^+ and the baseline method on the data sets are presented in Table 2. Since in classification of unbalanced data, area under ROC curve (AUC) and the balancing of the sensitivity and specificity are equally important, we evaluate the methods using a new evaluation metric called AUC balanced (AUCB), which is defined as follows:

$$\text{AUCB} = \frac{1}{2}(\text{sensitivity} + \text{specificity})(1 - |\text{sensitivity} - \text{specificity}|). \quad (12)$$

Moreover, we evaluate the methods in terms of AUC and F-score, which are widely used to evaluate the goodness of a classifier. From the results, we find that using the kNN classifier, RPMBGA^+ obtains better results in terms of AUC, AUCB and F-score than the baseline method on either of the data sets. On BP and TG data, RPMBGA^+ improves AUC, AUCB and F-score of baseline method by (15%, 14%, 14%) and (13%, 10%, 13%), respectively. When C4.5 is used as the classifier, RPMBGA^+ improves the results of baseline method on TG data set but it is no better than the baseline method on the blood pressure data set that has the larger number of features and is relatively more unbalanced than the TG data set. The probable reason may be that using information entropy, C4.5 selects those features that best classify the data, and if RPMBGA^+ picks up the same features, they both will produce the same classification statistics. When SVM with RBF/linear kernel is used as the classifier, either GA or RPMBGA^+ obtains very useless results; all the instances from the majority class are correctly classified but none or very few of the instances from the minority class are correctly classified, which returns the fitness score of 0. (Due to space limitation, detailed results are not presented.)

Since the kNN classifier obtains the better results than the C4.5 classifier, we analyze those factors that are picked by RPMBGA^+ as the important factors for the target health risk factor in various runs. From the the list of factor subsets selected by RPMBGA^+ as the best ones in various runs, we take the one that results in the highest fitness score as well as the highest AUCB. For TG, we have found a subset of 14 factors that result in the best fitness score; these factors include {SLEEPING TIME; FREQUENCY OF EATING SWEETS and SNACKS; HABIT OF EATING BETWEEN-MEAL AND MIDNIGHT SNACK;

Table 2. Classification statistics

Data	Method	Classifier	Sensitivity	Specificity	AUC	AUCB	F-score
BP	RPMBGA^+	kNN	0.68±0.02	0.63±0.02	0.66±0.01	0.63±0.02	0.44±0.05
		C4.5	0.37±0.03	0.85±0.02	0.61±0.02	0.32±0.03	0.38±0.03
	Baseline method	kNN	0.51±0.05	0.50±0.03	0.51±0.03	0.49±0.03	0.30±0.03
		C4.5	0.38±0.04	0.84±0.02	0.61±0.02	0.33±0.04	0.38±0.04
TG	RPMBGA^+	kNN	0.79±0.03	0.57±0.02	0.68±0.02	0.53±0.03	0.56±0.03
		C4.5	0.52±0.04	0.87±0.03	0.70±0.02	0.45±0.04	0.57±0.03
	Baseline method	kNN	0.65±0.03	0.44±0.03	0.55±0.02	0.43±0.02	0.43±0.03
		C4.5	0.51±0.02	0.82±0.02	0.66±0.01	0.46±0.03	0.52±0.02

BODY FAT RATE; TG OF THE LAST YEAR; AVERAGE TG OF PREVIOUS YEARS} and so forth. Interestingly, it has been observed that the two trivial factors {TG OF THE LAST YEAR; AVERAGE TG OF PREVIOUS YEARS} greatly affect the prediction rate of TG by RPMBGA⁺ and they are present in the best subsets of factors found in various runs. Similarly, taking blood pressure as the target health risk factor, we found a factor subset of 22 factors that results in the highest fitness score as well as the highest AUCB. These factors include {AGE; COMMUTING TIME; LIVING STATUS; HABIT OF WALKING, EATING FISH, EATING SNACKS, EATING BREAKFAST, DRINKING BEER, and DOING PHYSICAL EXERCISE; SLEEPING TIME; BODY FAT RATE; LOW BLOOD PRESSURE} and so forth.

4 Conclusion

In this paper, we propose a framework for prediction of health risk with high accuracy and identification of the factors that are highly related with the target health risk factor, such as blood pressure and TG. To evaluate the goodness of a feature subset for classification of unbalanced data, we have proposed three evaluation parameters; these parameters are utilized to determine whether to return the total misclassification cost or a score using sensitivity and specificity information. To utilize the known information about the relationships of the features with the target health risk factor, we propose a hybrid genetic algorithm that generates offspring by simultaneously using crossover and probabilistic sampling. However, using genetic algorithm does not produce a small informative feature subset. Instead, random probabilistic model building genetic algorithm+ (RPMBGA⁺), which generates offspring by the sampling probability vector of the features being selected, returns a feature subset with a very small number of features selected, which provide insights into the problem. Moreover, to deal with the variable length time series data like the health checkup and lifestyle data, we have proposed sliding window method in the kNN classification to calculate the distance and to predict the future health risks. By performing experiments on the health checkup and lifestyle data of Toshiba Corporation, we have found that the proposed method performs better than the baseline method in terms of area under ROC curve (AUC), the balanced AUC (AUCB), and F-score and selects a small number of important features.

However, there remain some unresolved issues that we want to address in our future work. For example, we want to perform additional experiments on the health checkup and lifestyle data, and on publicly available unbalanced data using our proposed methods and other classifiers.

References

1. MedlinePlus: Metabolic syndrome [Online accessed June 27, 2008] (2008), <http://www.nlm.nih.gov/medlineplus/metabolicsyndrome.html>
2. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading (1989)

3. Holland, J.: *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor (1975)
4. Koza, J.R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge (1992)
5. Paul, T.K., Iba, H.: Prediction of cancer class with majority voting genetic programming classifier using gene expression data. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* (23 August 2007); Preprint on IEEE Computer Society Digital Library, June 11 (2008)
6. Paul, T.K., Hasegawa, Y., Iba, H.: Classification of gene expression data by majority voting genetic programming classifier. In: *Proceedings of the 2006 IEEE WCCI, Vancouver, BC, Canada*, pp. 8690–8697 (2006)
7. Paul, T.K., Iba, H.: Gene selection for classification of cancers using probabilistic model building genetic algorithm. *BioSystems* 82(3), 208–225 (2005)
8. Paul, T.K., Iba, H.: Identification of informative genes for molecular classification using probabilistic model building genetic algorithm. In: *Proceedings of Genetic and Evolutionary Computation Conference 2004*, pp. 414–425 (2004)
9. Deb, K., Reddy, A.R.: Reliable classification of two-class cancer data using evolutionary algorithms. *BioSystems* 72, 111–129 (2003)
10. Wang, L., Chu, F., Xie, W.: Accurate cancer classification using expressions of very few genes. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 4(1) (2007)
11. Quinlan, J.: *C4.5: Programs for Machine Learning*. Morgan Kaufman Publishers, San Francisco (1993)
12. Vapnik, V.: *The Nature of Statistical Learning Theory*. Springer, New York (1995)
13. Tan, K.C., Tay, A., Lee, T.H., Heng, C.M.: Mining multiple comprehensible classification rules using genetic programming. In: *Proceedings of the 2002 Congress on Evolutionary Computation*, Washington, DC, USA, pp. 1302–1307 (2002)
14. Alfaro-Cid, E., Sharman, K., Esparcia-Alcázar, A.I.: A genetic programming approach for bankruptcy prediction using a highly unbalanced database. In: *Giacobini, M. (ed.) EvoWorkshops 2007. LNCS*, vol. 4448, pp. 169–178. Springer, Heidelberg (2007)
15. Pelikan, M., Goldberg, D., Lobo, F.: A survey of optimizations by building and using probabilistic models. Technical Report, Illigal Report 99018, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, USA (1999)
16. Paul, T.K., Ueno, K., Iwata, K., Hayashi, T., Honda, N.: Risk prediction and risk factors identification from imbalanced data with rpmbga+. In: *GECCO 2008: Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, pp. 2193–2198. ACM, New York (2008)
17. Baluja, S.: Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, Pennsylvania (1994)
18. Witten, I.H., Frank, E.: *Data Mining: Practical machine learning tools and techniques*, 2nd edn. Morgan Kaufmann, San Francisco (2005)

Extremal Optimisation and Bin Packing

Tim Hendtlass¹ and Marcus Randall²

¹ Faculty of Information and Communication Technologies
Swinburne University of Technology
Victoria Australia
`thendtlass@swin.edu.au`

² School of Information Technology
Bond University
Queensland Australia
`mrandall@bond.edu.au`

Abstract. Extremal Optimisation (EO) is a fairly new entrant into the realms of stochastic based optimisation techniques. Its behaviour differs from other more common algorithms as it alters a poorly performing part of the one solution used without regard to the effect this will have on the quality of the solution. While this means that its performance on assignment problems may be poor if used on its own, this same ‘failing’ makes it a very suitable base for a meta-heuristic. An analysis of the performance of naive EO on the classic bin packing problem is performed in this paper. Results are also presented that show that the same naive EO can be used in a meta-heuristic that performs very well.

1 Introduction

Extremal Optimisation (EO) is a fairly new entrant into the realms of stochastic based optimisation techniques¹. It owes much to the work of Bak-Sneppen [4,3] on self organised criticality. A common example of self organised criticality is the sand pile produced by slowly and gently dropping grains of sand onto a point. A pile builds but at various times the addition of one more grain results in a sand slide or avalanche, the frequency between avalanches being related to their magnitude. Large avalanches are exponentially less likely to occur than smaller ones. This relationship is found in other naturally occurring phenomena too, the Gutenberg-Richter Law [2] for example stating that earthquakes of magnitude n will occur about ten times as frequently as a earthquake of magnitude $n + 1$.

Self organised criticality has been postulated for sets of interacting species and used to explain the sudden disappearance of some species. In this model a less fit species S might disappear from time to time, with effects on other species above and below S in the food chain. For the purpose of this model the vanishing species are replaced with randomly generated replacement species. From time to time an avalanche will occur with a number of species disappearing at once,

¹ For an introduction to the mechanics of EO, see Randall [10].

again with an exponential inverse relationship between the number of species vanishing and the frequency of such an event.

EO attempts to apply these ideas to a solution to some problem when that solution is composed of various elements. No doubt the worth of the solution is some function of both the element values and how these interact. If the weakest element can be identified (see Section 2 for two examples how this could be done) and altered, self organised criticality will allow for continual small changes in overall fitness with some sudden larger changes (again at a frequency inversely related to their magnitude). Big changes might dramatically decrease the fitness but the slow non-monotonic improvement that follows will be occurring in a different part of problem space than that that was being explored before the avalanche. Keeping a record of the best place or places explored by this single changing solution as it explores problem space may provide a range of good solutions to the original problem.

The original approach to always change the worst element proved to be too greedy for real use and so weighted choice was introduced that would on average pick the worst element of the N available but might choose another. Elements were rated from the best to the worst ($1 \dots N$). Then the probability that the element with rank R would be chosen is given by $P_R = \left(\frac{R}{0.5N(N+1)} \right)^\tau$. τ is the only user chosen parameter for EO and for the work reported in this paper has been set to 1.4 following the recommendations in [8,6,7].

In this paper bin packing is used to explore how EO can operate as an optimisation algorithm, both on its own and as a component of a meta-heuristic.

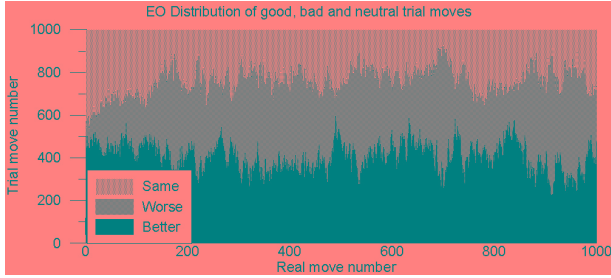
2 EO and Bin Packing

In bin packing the objective is to distribute N objects of different but fixed weight between B bins each with a definite capacity so that the sum of the excess weight in all the bins is as low as possible, ideally zero. EO considers the bins to be the elements of the solution; at each iteration it chooses one of the overfull bins probabilistically using the formula above and then randomly takes one item from that bin and places it in another bin. In this type of problem moving one item only affects the two bins involved, that is changing one element has only a slight effect on the rest of the elements that make up the solution.

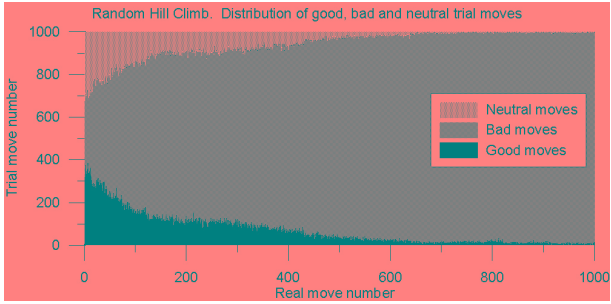
The bin packing problem used to generate the results in the next section is u120_00 [5] which involves 48 bins each with a maximum capacity of 150 units. 120 items ranging in size from 20 to 98 units and with a total size of 5735 units are to be distributed between these bins. The maximum overfull amount would be 5585 (if all items were in one bin) and the minimum zero.

3 The Performance of Basic EO on Bin Packing

Figure 1 shows the results from an EO run in which 1000 exploratory moves were made (and discarded) starting from each real move the algorithm made.



(a) Relative abundance of good, bad and neutral moves at each of the 1000 actual moves for EO.



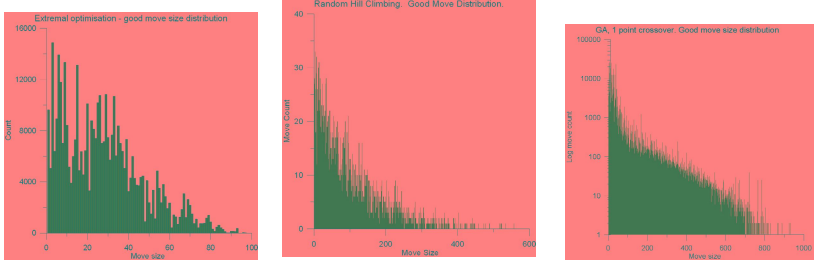
(b) Relative abundance of good, bad and neutral moves at each of the 1000 actual moves for RHC.



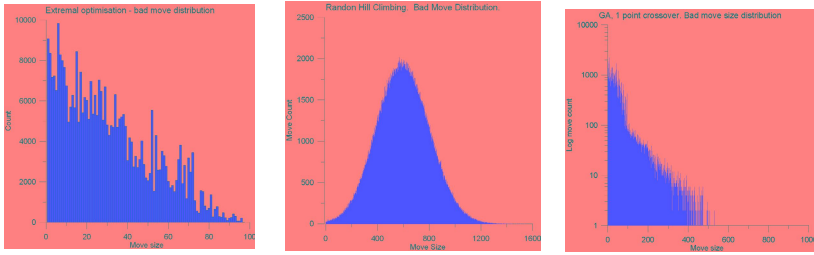
(c) Relative abundance of good, bad and neutral moves at each of the 1000 actual moves for GA.

Fig. 1. Histograms of the quality of the 1000 trial moves carried out at each real move position on bin packing (problem u120_00) for each of the three algorithms

Figure 1(a) shows the relative abundance of good moves (ones to a solution with less constraint violations than the current solution), bad moves (ones with more constraint violations) and neutral moves (ones with the same number of constraint violations).



(a) EO Good trial moves. (b) RHC Good trial moves. (c) GA Good trial moves.



(d) EO Bad trial moves. (e) RHC Bad trial moves. (f) GA Bad trial moves.

Fig. 2. Histograms of the quality of the 1000 trial moves carried out at each real move position on bin packing (problem u120_00) using each of the three algorithms. Note that the vertical axis for the GA results are plotted to a log scale while the results for the other two algorithms are plotted to a linear scale.

This information, while of interest in itself, can be put into perspective by a comparison with the equivalent information for two other algorithms applied to the same problem. The first of the algorithms (Figure 1(b)) is a Random Hill Climber (RHC). This also uses a single solution, and each real move is made by generating a new random solution and, if it is better than the current solution, replacing the current solution with this new solution. This algorithm can be made less greedy if a single parameter is introduced in the interval from $[0.5 \dots 1]$ which is interpreted as the probability that a better trial solution replaces the current solution. This, like EO, uses a single solution and has one user chosen parameter, but works using a conditional pull toward a better solution unlike EO which pushes away from the current solution in a way that might be better or might be worse. The results shown in Figure 1(b) again include the statistics built from performing (and reversing) 1000 trial moves for each real move attempt. The second algorithm used for comparison purposes is a genetic algorithm (GA) (Figure 1(c)). This uses a population of solutions (20 for these results) and forms new solutions from two parent solutions. Parent solutions are found using tournament selection between pairs of randomly chosen solutions with a probability of taking the fitter (least constraint violations) being 0.7. A

new solution is built from the two parents using single point crossover and there is a probability of 0.1 that one randomly chosen item in the child is moved to a different randomly chosen bin. Continuous replacement is used in which the new child replaces the least fit solution in the population if the new solution is fitter. This is almost certainly not the ideal GA if the intent is to solve the problem, but is chosen to be most similar to the structure of the other two algorithms (given that this is a population based algorithm and the other two only use a single solution) so that the results can be most readily compared. The 1000 trial moves for each real move create new solutions in the same way as for real moves, but are discarded as soon as the fitness of the new solutions is known.

A comparison of the relative availability of the good, bad and neutral moves available to the three algorithms in Figure 1 is informative. The Random Hill Climber, as might be expected from its very greedy implementation, rapidly runs out of good moves. This corresponds to becoming stuck in a local optimum with better solutions (fewer constraint violations) being so rare as not to be found among the random solutions generated during the remainder of the run. The relationship between solution quality and the number of solutions with this quality of result is such that a change by a factor of two in the result quality means a change of orders of magnitude in the number of solutions with this new quality. The GA proceeds in a series of breakthroughs, punctuated by regions of relative stability in which few good moves are available to the algorithm. Immediately after a breakthrough a number of good moves are available but are soon exhausted. The EO algorithm always has a significant number of good moves available to it, and the number of good to bad moves is approximately equal.

The relative abundance of moves does not give the full picture; how good and how bad these moves are is also important. Figure 2 gives histograms of the good and bad move sizes for each of the algorithms. Note that for EO the histograms are similar, except for the larger number of small good moves *c.f.* small bad ones. As might be expected for an algorithm that generates random solutions, the bad solutions that the RHC generates show a roughly normal distribution with quite a high central value. The good moves for this algorithm are biased towards small moves. Since the GA is building from known, if not good, reasonable solutions, there are more and better good than bad moves available.

The ratio of good moves to bad becomes important for any algorithm which, like EO, makes moves no matter whether this will improve the solution quality or not. With an average good to bad ratio of 1.17 to 1 the probability that EO alone will put together a series of improving moves necessary to reach a good quality result is very small. Random Hill Climbing has a good move / bad move ration of 0.08 to 1, but since new solutions are only accepted if they are better, while the time between new moves being accepted may be large the final result may still be good given a long enough time. A GA already has a good to bad ratio of 5.7 to 1 and also only accepts new solutions into the population if better than the worst performing population member. This makes it a clear choice of the three algorithms described here if a good solution must be found

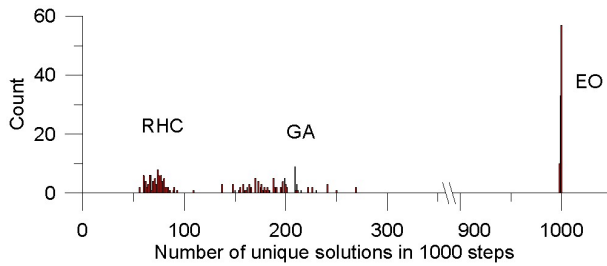


Fig. 3. The number of unique solutions explored during the 1000 moves of each algorithm

fast. However, a GA may easily become trapped in a local optimum which may or may not be adequate for purpose.

This paper, however, looks at EO as a basis of a meta-heuristic and for this role the apparent disadvantage described above becomes an advantage. Consider Figure 3 which shows the number of unique solutions explored during 100 repeat runs, each of 1000 steps, by each of the three algorithms (without any trial moves). The unconditional acceptance of new solutions by EO results in very little re-exploration, unlike the other two algorithms. Uniqueness of the solutions explored is a necessary but not sufficient condition for the basis of a good meta-heuristic. If one just needed low re-exploration, randomly generated solutions would be very hard to beat. However, the quality of these solutions is also important. EO with an average fitness of 783 is clearly better than the randomly generated solutions with an average fitness of 1833. As well as the overall average the change in quality with time is also of interest. Figure 4 shows that, as might be expected, quality just fluctuates around a mean value for randomly generated solutions. EO, however, tends to improve as the run progresses, although far from monotonically.

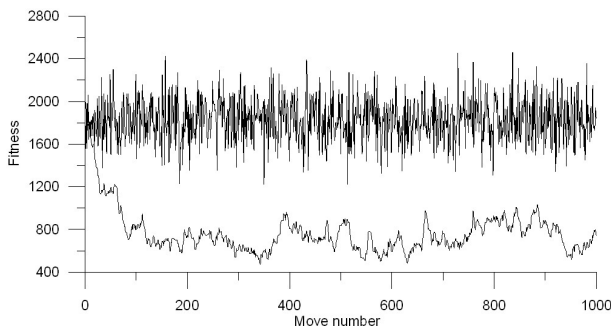


Fig. 4. The fitness of the single solution during 1000 steps of the Extremal Optimisation algorithm (lower trace) compared with 1000 randomly chosen solutions. Note the upper trace is not that of a random hill climber.

The results from this section indicate that EO on its own will probably not perform well on bin packing problems (as least not in a short time). If basic EO can be augmented by additional local search techniques the resultant meta-heuristic may be expected to outperform both basic EO and other techniques that rely on multiple runs starting from random positions.

4 The Performance of EO Plus Enhancements on Bin Packing

While EO has the ability to move through search space without being trapped in local optima, its ability to refine solutions is rather limited. To this end, like other meta-heuristics, local search becomes a necessity. Two levels of local refinement, a general purpose and a specific heuristic, can be added to naive EO to produce a good meta-heuristic for bin packing.

- *General Purpose* - The heuristic known as *partial feasibility restoration* by Randall [10] can be adapted to reduce the number of constraint violations. It is a simple, non-degenerative, parameter-free process. In the case of bin packing it works as follows. Each bin is examined in turn, and if found to be overloaded, the item that most closely matches the surplus amount is identified. If another bin can be found to accommodate this item, it will be added to it, else no action is taken. It is an $O(BN)$ algorithm where B is the number of bins, and N is the number of items.
- *Specific Heuristic* - A modified version of the algorithm of Alvim, Aloise, Glover and Ribeiro [1] (as reported in Levine and Ducatelle [9]) is used to minimise the collective excess weight of the bins. The original algorithm also attempted to remove bins as well. Our version initially determines the two least loaded bins. The items from these are moved to a “free list”, and the bins (temporarily) are removed. Three types of exchange operations are attempted. In the first, two items from the free list are exchanged with two items from a bin (subject to the capacity constraint being satisfied). This process is repeated for all bins and this is attempted for combinations of two bin items for one free item and then one bin item for one free item. After this, all free items are distributed to the empty and non-empty bins. In essence, it is able to better pack the least loaded bins and thus performs a useful redistribution function. A more detailed description of the algorithm (as well as an example) can be found in Levine and Ducatelle [9].

Given that two local search heuristics can be applied to solutions that EO generates, four combinations of search strategy are possible. To demonstrate the effectiveness of these heuristics, the four possibilities are applied to the problem u120.00. Additionally, these results are contrasted with a random start procedure that simply generates a random solution at each iteration. The results are given in Table 1.

A number of conclusions can be drawn from Table 1. Although EO by itself is not an effective optimiser on the problem, when the special purpose heuristic

Table 1. All combinations of heuristic strategies on u120_00. Note that “Iteration” refers to the iteration at which the best solution cost was found. The maximum number of iterations was 100000. “Min”, “Med” and “Max” denote minimum, median and maximum respectively.

Meta-heuristic	Heuristic		Cost			Iteration		
	Base	General Special	Min	Med	Max	Min	Med	Max
EO	No	No	66	101	143	14734	47451	81805
	No	Yes	0	0	0	2676	3487	4522
	Yes	No	78	90.5	101	4696	55587	93606
Random Start	Yes	Yes	0	0	0	9	36.5	179
	No	No	966	1076.5	1132	18009	56566	97472
	No	Yes	966	1076.5	1132	18009	56566	97472
	Yes	No	214	251.5	272	30305	41144.5	98881
	Yes	Yes	214	251.5	272	30305	41144.5	98881

Table 2. Fifteen problem instances of varying size are tested on EO with and without both types of local search

		Local Search Off						Local Search On					
Instance	Bins	Cost			Iteration			Cost			Iteration		
		Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max
u120_00	48	66	101	143	14734	47451	81805	0	0	0	12	32.5	95
u120_01	49	48	83.5	122	26065	64875.5	97475	0	0	0	6	10	29
u120_02	46	50	78	98	3816	31577	90114	0	0	0	10	21	36
u120_03	49	92	129	177	16630	64697	92822	0	0.5	19	36	256	1267
u120_04	50	38	61.5	99	10641	50991.5	93628	0	0	7	14	20.5	152
u250_00	99	284	311.5	335	6967	34316.5	97497	0	11.5	15	50	323	8386
u250_01	100	203	254.5	305	13557	38145.5	90769	0	0	12	22	241	612
u250_02	102	271	315.5	352	7189	35449.5	84125	0	0	10	145	265	10296
u250_03	100	261	278	328	8779	43386	93587	0	0	4	21	48.5	1103
u250_04	101	294	349.5	404	12194	53111	97232	0	7	22	55	494.5	9628
u500_00	198	619	657	705	26501	72795.5	83610	26	40.5	65	8671	56455.5	90245
u500_01	201	671	731	778	21657	62828	97751	22	36	56	1371	30558.5	76010
u500_02	202	620	717	750	31921	90804	99027	2	7	25	288	3267.5	48277
u500_03	204	665	738.5	767	20540	73154.5	85891	26	43.5	61	1078	46085.5	96376
u500_04	206	586	677.5	742	22244	53668.5	95949	14	18.5	34	2928	4246	71808

is applied, it is able to pack the items into the required 48 bins. It is able to do this in far fewer iterations if the general purpose heuristic, partial feasibility restoration, is used as well. The random start procedure, despite the aid of the two heuristics, is unable to generate the quality of solutions of EO. Interestingly, the special purpose heuristic has no effect on the solutions generated by random start, whereas partial feasibility restoration does. It is likely that only EO is able to provide sufficiently refined solutions for this special purpose heuristic to work.

To demonstrate the behaviour of EO more fully, Table 2 gives the results of fifteen bin packing instances, ranging in size from 120 items to 500 items. Here EO is used with and without the two heuristics. The results clearly show that EO coupled with these two heuristics is generally quite effective on bin packing problems. While EO was not able to solve the 500 item problems exactly, subsequent testing revealed that it was able to generate feasible solutions if the number of bins was increased by one.

5 Concluding Comments

All the results quoted so far refer only to bin packing problems. A full set of results has been collected for graph colouring problems² as well but cannot be reproduced owing to space constraints. For all these problems, while exact values may differ, the same general trends can be observed giving confidence that the conclusions drawn are not a result of the class of problem chosen but may be considered generally applicable across a range of assignment problems.

References

1. Alvim, A., Aloise, D., Glover, F., Ribeiro, C.: Local search for the bin packing problem. In: *Extended Abstracts of the Third Metaheuristics International Conference*, pp. 7–12 (1999)
2. Bak, P.: *How Nature Works*. Springer, Heidelberg (1996)
3. Bak, P., Sneppen, K.: Punctuated equilibrium and criticality in a simple model of evolution. *Physical Review Letters* 71, 4083–4086 (1993)
4. Bak, P., Tang, C., Wiesenfeld, K.: Self-organized criticality: an explanation of 1/f noise. *Physical Review Letters* 59, 381–384 (1987)
5. Beasley, J.: OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society* 41, 1069–1072 (1990)
6. Boettcher, S.: Extremal optimization of graph partitioning at the percolation threshold. *Journal of Physics A: Mathematical and General* 86, 5201–5211 (1999)
7. Boettcher, S., Percus, A.G.: Extremal optimization for graph partitioning. *Physical Review E* 64, 26–114 (2001)
8. Boettcher, S., Percus, A.G.: Optimization with extremal dynamics. *Physical Review Letters* 86, 5211–5214 (2001)
9. Levine, J., Ducatelle, F.: Ant colony optimisation and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society* 55, 705–716 (2004)
10. Randall, M.: Enhancements to extremal optimisation for generalised assignment. In: Randall, M., Abbass, H., Wiles, J. (eds.) *ACAL 2007*. LNCS, vol. 4828, pp. 369–380. Springer, Heidelberg (2007)

² In graph colouring the object is to assign colours to nodes in such a way that, ideally, no two connected nodes have the same colour. Nodes are connected by links, but not all nodes are connected to all other nodes. The fitness is the number of links that have the same colour in the nodes at each end, ideally zero. EO considers the nodes to be the elements of the solution and notes how many colour conflicts each node is involved in. At each iteration it probabilistically chooses one of the nodes involved in these conflicts (using the formula above) and then changes the colour of that node to a random colour. In this type of problem changing the colour of one node to another colour will not only affect the score of that node changed but may also affect the scores of all the other nodes having links with this node. Now changing one element may have major effects on the rest of the elements that make up the solution.

Extremal Optimisation with a Penalty Approach for the Multidimensional Knapsack Problem

Pedro Gómez-Meneses^{1,2} and Marcus Randall¹

¹ School of Information Technology, Bond University, QLD 4229, Australia
`pedgomez@bond.edu.au`, `mrandall@bond.edu.au`

² Universidad Católica de la Santísima Concepción, Concepción, Chile
`pgomez@ucsc.cl`

Abstract. The extremal optimisation (EO) meta-heuristic is a recent form of search that is suitable for combinatorial optimisation problems. EO has been applied to problems such as graph partitioning, spin glass, and graph colouring. However, only a relatively small amount of work has been done on other combinatorial problems particularly those having constraints. This paper examines the issue of satisfying constraints with a penalty approach using the multidimensional knapsack problem. An EO model is presented which finds solutions through the analysis of the number of overloaded constraints. This approach allows the solution state move between feasible and infeasible spaces. The results show that the new algorithm is able to obtain optimal results for small problems and finds competitive solutions for large problems.

1 Introduction

A number of recent meta-heuristic search algorithms are inspired by nature's ability to adapt and evolve to suit given environments. EO is a novel evolutionary method of optimisation proposed by Boettcher and Percus [1], founded on the self organising criticality (SOC) [2] paradigm of co-evolution introduced by Bak and Sneppen [3]. EO has been applied to problems such as the graph bi-partitioning, 3-colouring graph, spin-glass, and max-cut [4, 5]. Some exploratory work on other combinatorial problems such as the travelling salesman [1, 6], multidimensional knapsack (MKP) [6], maximum satisfiability [7], generalised assignment [8], and dynamic problems [9] has also been undertaken. Hybrid algorithms have also been developed in which aspects of EO are combined with other methods such as particle swarm optimisation, and genetic algorithms (GAs) [10, 11].

This paper proposes the use of a novel criterion to define the fitness of each solution element with a penalty approach. The test problem that we use to demonstrate this is the MKP. The selection criterion to define the fitness of each knapsack item is based on the number of restrictions that are violated by the item together with the profit that each item contributes to the maximisation of the objective function. The advantage of this technique is its simplicity of implementation compared with other proposals based on the combined use of linear programming methods and evolutionary methods [12, 13, 14, 15].

Given this new approach, we test the behaviour of the EO model on the MKP through well-known benchmark problems from the OR-Library [16]. The preliminary results obtained show that the proposed model is an effective approach and competitive with other techniques. In order to further improve these results, also a simple local search mechanism based on items exchange was implemented.

The paper is organised as follows. Section 2 gives a summary of the EO model. Section 3 explains how the proposed EO model is applied to MKP. Section 4 shows the results obtained and presents an analysis of them. Finally in Section 5 we conclude and discuss the future work arising from this study.

2 Extremal Optimisation

EO is based on the Bak-Sneppen model which employs some components of SOC [2]. SOC is used to describe the behaviour of dynamic systems that have the capacity to reach a balanced state by themselves until a new event of major magnitude destabilises it. The frequency with which these events occur follows a power law distribution. Classical examples of this are the formation of sand piles and the occurrences of earthquakes [17].

The Bak-Sneppen model considers the species as the minimum unit of fitness evaluation. Species are influenced by the fitness of their nearest neighbouring species in the landscape in order to form the co-evolutionary ecosystem.

Unlike other meta-heuristics, EO is based on eliminating the elements that degrade the system instead of selecting the better elements for the next generation. The latter technique is commonly used by other evolutionary algorithms.

The original EO algorithm always eliminates the element with the worst fitness. This action causes the algorithm to have a deterministic behaviour with a high probability of falling into local optima. For this reason, Boettcher and Percus [18] proposed a new version called τ -Extremal Optimisation (τ -EO). τ -EO¹ improves results and increases the likelihood of escaping from locally optimal solutions simply by establishing a new parameter τ that permits a probabilistic choice of the element to be eliminated rather than necessarily the worst. The modification evaluates and ranks the fitness of all elements from the poorest to the best and a rank between from 1 and n is assigned to them respectively. Equation 1 shows how to calculate the probability for each ranked element.

$$P_i = i^{-\tau} \quad \forall i \quad 1 \leq i \leq n, \quad \tau \geq 0 \quad (1)$$

where:

n is the total number of elements evaluated and ranked, and

P_i is the probability that the i^{th} element is chosen.

Roulette wheel selection (or similar) is used to choose the element² whose value is to change. If new solution is the best found so far, according to the cost function, then X_{best} is updated. This method, shown in Algorithm 1, requires very few run-time parameters and its implementation is simple.

¹ From hereon in, the term EO is used interchangeably with τ -EO.

² Hereinafter, the term “item” will be used instead of “element”.

Algorithm 1. Standard EO pseudo-code for minimisation problem

Generate an initial random solution $X=(x_1, x_2, \dots, x_n)$ and set $X_{\text{best}} = X$;
for a preset number of iterations **do**
 Evaluate fitness λ_i for each variable x_i , $1 \leq i \leq n$;
 Sort fitness λ_i for each x_i from worst to best;
 Generate a random variate $P_{rand} \sim U(1, 0)$;
 Select an item x_j using the roulette wheel method and the P_{rand} probability;
 Choose X' in the neighbourhood of X where x_j 's value changes to a random one;
 New solution $X = X'$ is accepted *unconditionally*;
 Evaluate the cost function $C(X)$;
 if $C(X) < C(X_{\text{best}})$ **then**
 Set $X_{\text{best}} = X$;
 end if
end for
Solution X_{best} and $C(X_{\text{best}})$ are returned;

3 EO for the MKP

3.1 Multidimensional Knapsack Problem

The MKP is an $\mathcal{NP}$ -hard combinatorial optimisation problem and is a generalisation of the knapsack problem which has only one constraint. The aim of MKP is to find a combination of n items that can maximise profit without exceeding the amount of resources allowed for the m constraints. Formally, it can be represented as:

$$\text{Maximise} \quad \sum_{i=1}^n p_i x_i \quad (2)$$

$$\text{Subject to} \quad \sum_{i=1}^n r_{ij} x_i \leq b_j \quad \forall j \quad 1 \leq j \leq m \quad (3)$$

$$x_i \in \{0, 1\} \quad \forall i \quad 1 \leq i \leq n \quad (4)$$

where:

- p_i is the profit of item i ,
- x_i is the decision variable, 1 if the item is in the solution, 0 otherwise,
- r_{ij} is the coefficient or weight of item i in constraint j , and
- b_j is the boundary of the constraint j .

MKP can be applied to problems such as the capital budgeting problem, the cargo loading problem, the cutting stock problem, and the processor allocation problem in distributed systems [15]. It is also widely used as a benchmark problem to compare new general purpose meta-heuristics [12, 13, 14].

3.2 An EO implementation for the MKP

Many implementations to solve MKP use complex methods to ensure that all solutions are feasible [12, 15], or use GAs with penalty functions to solve constrained problems [19, 20, 21]. According to Randall [8], EO is an appropriate

method for problems with constraints as it can naturally move between feasible and infeasible space. Thus, we choose to use a separate fitness evaluation with a novel penalty function which depends on whether the current solution is feasible or infeasible. Equation 5 shows the common penalty function.

$$\lambda_i = \begin{cases} f(x_i) & \text{if the solution is feasible} \\ f(x_i) + pf(x_i) & \text{if the solution is infeasible} \end{cases} \quad (5)$$

where:

- λ_i is the fitness value of item i ,
- $f(x_i)$ is item i 's contribution to the objective function,
- $pf(x_i)$ is the penalty function evaluated for item i .

Alternatively, we propose to apply a penalty function to evaluate the fitness of an item in the complete search space. That is, if the current solution is feasible then new infeasible solutions from this are penalised. On the other hand, if the current solution is infeasible then the new solutions that are further off in infeasible space are penalised. Equation 6 shows the novel fitness evaluation.

$$\lambda_i = \begin{cases} pf_{[fs]}(x_i) & \text{if solution is feasible} \\ pf_{[is]}(x_i) & \text{if solution is infeasible} \end{cases} \quad (6)$$

where:

- $pf_{[fs]}(x_i)$ is the penalty function evaluated for item i for feasible solutions,
- $pf_{[is]}(x_i)$ is the penalty function evaluated for item i for infeasible solutions.

In the penalty functions $pf_{[is]}(x_i)$ only items with a value of 1 are evaluated, items with a value of 0 do not take part in this process of evaluation. First, the number of constraints that remain overloaded (R) for the current solution is determined. Then, for all $x_i = 1$, let R_i be the number of constraints that would be violated if x_i is set to 0 and all other x_j 's remain unchanged; that is, it is taken out of the knapsack. If R_i reduces its value, this means that the item overloads the knapsack because, without it, the number of overloaded constraints is less than the current number of R . Thus, items with lower values of R are good candidates to be changed. The profit value p of the objective function is used as second criterion to rank the items when two or more items have the same value for their respective R 's. The profit value p_i for each item i is transformed to a normalised p'_i between 0 and 1 using Equation 7.

$$p'_i = \left(\frac{p_i}{p_{max} + 1} \right) \quad \forall i \quad 1 \leq i \leq m \quad (7)$$

where:

- p_{max} is the profit with the greatest value of all items,
- m is the number of items to be normalised, and
- p'_i is the normalised profit of the i^{th} item.

An item with a low p_i value is a good candidate to be taken out because this has the least effect on the value of the solution when the objective function must be maximised. The proposed method is based on the fitness representation form in which the integer part represents the number of overloaded constraints and the decimal part represents the normalised profit value. Thus, in Equation 8 the fitness λ_i for each item is given by the sum of R_i and p'_i and items with lower fitness are candidates to be taken out.

$$\lambda_i = R_i + p'_i \quad \forall i \text{ such that } x_i = 1 \quad (8)$$

where:

R_i is the number of overloaded constraints for the i^{th} item, and
 λ_i is the fitness of the i^{th} item.

In the penalty functions $pf_{[fs]}(x_i)$ only items with a value of 0 are evaluated, items with a value of 1 do not take part in this process of evaluation. First, the number of constraints that could be overloaded, C_i , for each item i , is calculated when x_i changes from 0 to 1; that is, it is put back into the knapsack. A C_i with a value greater than 0 means that the item is an element that overloads one or more of the constraints. Therefore, the items with a value equal to 0 or with a very low value of C are good candidates to be incorporated. In the same way as the case for infeasible solutions, the profit value p is used as second criterion to rank the items that have the same C value (see Equation 7). An item i with a high value of p_i is a good candidate to put in because this helps to increase the value of the objective function. Following the mechanism of choosing the least value of fitness, the profit is represented by its complementary value $1 - p'_i$. Thus, in Equation 9 the fitness λ_i for each item is given by the sum of C_i and $(1 - p'_i)$ and items with lower fitness are candidates to be put in the knapsack.

$$\lambda_i = C_i + (1 - p'_i) \quad \forall i \text{ such that } x_i = 0 \quad (9)$$

where:

C_i is the number of overloaded constraints for the i^{th} item, and
 λ_i is the fitness of the i^{th} item.

If the selected item has a C_i value greater than 0, the new solution becomes infeasible. Thus, when a solution is infeasible, the new solution can become feasible or can remain infeasible. If the new solution is still infeasible, the same procedure will be applied; thus, the subsequent solutions will move toward feasible space. On the other hand, if the current solution is feasible the new solution can remain feasible or can become infeasible.

The proposed penalty approach does not require any additional parameters and can be applied to any combinatorial constrained problems without the necessity of specific knowledge of a particular problem.

Algorithm 2 calculates and analyses the fitness value for the MKP according to whether the current solution is feasible or infeasible. Thus, this algorithm has the characteristic that it never takes out an item if the current solution is feasible, and never adds an item if the current solution is infeasible.

4 Computational Experiments

The proposed τ -EO method was coded in the C language and compiled with **gcc** version 4.1.2. The computing platform used to perform the tests has a 1.86 GHz Intel Core2 CPU and 917 MB of memory, and runs under Linux.

The first part of the experiments applied the proposed EO model to solve a group of six small MKPs from OR-Library [16]. These problems, given in Table 1, were the same used by Randall and Lewis [6]. They implemented the first EO approach to solve the MKP. Initially, the standard EO found only one optimal value for the problem mknap1. The results obtained for the remaining

Algorithm 2. Pseudocode of the EO model for the MKP

```
Initialise a solution  $S$  with random values from  $\{0, 1\}$ ;  
if  $S$  is feasible then  $S_{best} \leftarrow S$ ;  
for a preset number of iterations do  
  if current solution is feasible then  
    Evaluate the fitness  $\lambda_i$  according to Equation 8;  
  else  
    Evaluate the fitness  $\lambda_i$  according to Equation 9;  
  end if  
  Rank items from lowest to highest fitness;  
  Generate a random variate  $P_{rand} \sim U(1, 0)$ ;  
  Choose an item using the roulette wheel method and the  $P_{rand}$  probability;  
  if current solution is feasible then  
    Change the value of the item chosen from 1 to 0;  
  else  
    Change the value of the item chosen from 0 to 1;  
  end if  
  Accept the new solution  $S_{new}$  unconditionally;  
  Evaluate the cost function  $C(S_{new})$  for the new solution;  
  if  $S_{new}$  is feasible and  $C(S_{new}) > C(S_{best})$  then  
     $S_{best} \leftarrow S_{new}$ ;  
  end if  
end for
```

problems were close to the optimal values. The time taken to find the best and mean result was measured by the number of iterations required to reach it. After that, the Extended Extremal Optimisation Model (EEO) proposed by Randall and Lewis [6] was applied to determine if better results could be reached. The results for the problems mknep3 to mknep5 were better but these did not reach the optimal value, and for the problem mknep6 the result was slightly worse.

Following Randall and Lewis [6], the number of iterations was set to 500000 to run the penalty-based EO method under similar conditions. However, the τ value was not specified explicitly. For this reason, we ran the small MKP for values of τ from 1.1 to 1.7. For values of τ greater than 1.7 the results obtained were further away from the optimal value. The value of τ that obtained the best result was 1.4. Each

Table 1. Comparative table of test results for different EO models

Problem			Standard EO				EEO	Penalty-based EO				
Name	N	M	Optimal	Best	Number of	Mean	Number of	Best	Best	Number of	Mean	Number of
			result	result	iterations	result	iterations	result	iterations	result	iterations	
mknnap1	06	10	3800	3800	9140	3800	9140	3800	3800	14	3800	14
mknnap2	20	10	6120	6040	166304	5980	70808	6040	6120	117609	6120	117609
mknnap3	28	10	12400	12180	161089	12180	161089	12210	12400	108981	12400	108981
mknnap4	39	05	10618	9083	287120	9022	197651	10052	10604	196769	10563	248079
mknnap5	50	05	6339	6097	172585	6003	31215	6107	6339	4029	6339	4029
mknnap6	60	05	6954	6650	119523	6583	169988	6570	6954	1926	6954	1926

problem was run 10 times. The best and mean results obtained with the number of iterations necessary to reach these results are shown in Table 1.

The results obtained show that the proposed EO method found 5 out of 6 optimal values for each of the ten times that the problems were run. Only for the problem *mnknap4* was the optimal value not found. However, the percentage gap between the optimal result and the best result found for the problem *mnknap4* was only 0.13%. Furthermore, the number of iterations to reach the best results was less than that of the EO model by Randall and Lewis [6]. Thus, for small problems, the modified EO proposed has proved to be a simple and faster mechanism to solve the MKP.

In the second part, we applied EO to solve 270 large MKPs created and solved by Chu and Beasley [12] which are available on the OR-Library [16]. These problems are grouped by the number of variables ($n \in \{100, 250, 500\}$) and the number of constraints ($m \in \{5, 10, 30\}$) making up 9 groups of 30 problems each. These 30 problems are subdivided, at the same time, into three subgroups of 10 problems by tightness ratio ($\alpha \in \{0.25, 0.5, 0.75\}$) of the resource constraints which was set using $b_j = \alpha \sum_{i=1}^n r_{ij}$.

The number of iterations again was set to 500000. The τ value was calculated through an empirical test with a reduced number of iterations. The test mechanism consisted of measuring the result of the average percentage gap obtained by the LP relaxation and EO, that is, $\%gap = 100 * (opt^{LP} - opt^{EO}) / opt^{LP}$. Thus, the τ value with the least percentage gap was chosen. τ was tested within the range of values between 1.8 and 2.8. For values outside this range of τ the results were worse. The value selected was $\tau = 2.2$.

Each problem was run ten times. The results were grouped by tightness ratio and the values shown in Table 2 are the averages of the results obtained for each group. Since the optimal solution values for these problems are not known, our results are compared with those obtained by Chu and Beasley (CB-GA) [12], Raidl (R-GA) [14], Uyar and Eryigit (UE-GA) [21], Gottlieb (G-EA) [13], and Vasquez, Hao and Vimont (VH-TS) [15, 22]. The average $\%gap$ with respect to the LP relaxed optimum is used to compare the results obtained for each method. Table 2 shows the values found and reported ³ by each method with the average of the ten best values found for each tightness group and/or the average percentage gap with respect to the LP method.

The results obtained show that EO found values close to those previously obtained by other more complex methods. The CB-GA method uses a repair operator that works with a surrogate relaxation technique. R-GA is based on an pre-optimisation of the initial population through an LP relaxation technique, a repair operator, and a local optimisation operator. The UE-GA method uses a gene-based adaptive mutation approach and six parameters that must be set. The G-EA method uses a decoder technique based on a permutation representation for the crossover and mutation operator and five parameters must be set. The VV-TS combines LP relaxation with tabu search which is used as a local search mechanism.

³ Blank spaces are present because that information was not provided in the paper.

Table 2. Comparative EO's test results. LS indicates the added procedure of local search. The local search was run after than the best solution by EO was found.

Problem			LP [12]	CB-GA [12]		R-GA [14]		UE-GA [21]	G-EA [13]	VV-TS [15, 22]		EO		EO+LS	
<i>m</i>	<i>n</i>	α	Average optimal values	Average best solution	Average % gap	Average % gap	Average % gap	Average % gap	Average % gap	Average best solution	Average % gap	Average best solution	Average % gap	Average best solution	Average % gap
5	100	0.25	24438.39	24197.20	0.99							24153.10	1.17	24153.10	1.17
		0.50	43449.50	43252.90	0.45						43113.50	0.77	43120.30	0.76	
		0.75	60663.76	60471.00	0.32						60434.30	0.38	60454.80	0.34	
		Average			0.59	0.59			0.59			0.77		0.76	
5	250	0.25	60547.41	60409.70	0.23					60410.5	0.23	59739.60	1.33	59795.60	1.24
		0.50	109411.71	109284.60	0.12					109292.8	0.11	108099.30	1.20	108239.50	1.07
		0.75	151676.33	151555.90	0.08					151560.3	0.08	150853.50	0.54	150967.90	0.47
		Average			0.14	0.15			0.17		0.14		1.02		0.93
5	500	0.25	120717.02	120615.50	0.09		0.59			120630.30	0.07	119041.80	1.39	119210.60	1.25
		0.50	219595.79	219503.10	0.04		0.21			219512.70	0.04	216456.70	1.43	216783.10	1.28
		0.75	302434.90	302354.90	0.03		0.09			302363.40	0.02	300603.20	0.61	300963.90	0.49
		Average			0.05	0.04	0.30	0.10		0.04			1.14		1.00
10	100	0.25	22960.49	22601.90	1.56							22537.70	1.84	22537.70	1.84
		0.50	43000.79	42659.10	0.79							42525.40	1.11	42525.50	1.11
		0.75	59844.23	59555.60	0.48							59488.70	0.59	59490.10	0.59
		Average			0.94	0.95			0.97				1.18		1.18
10	250	0.25	59290.15	58993.90	0.51					59021.60	0.45	58407.60	1.49	58468.00	1.39
		0.50	108980.95	108706.40	0.25					108729.30	0.23	107701.70	1.17	107835.70	1.05
		0.75	151560.07	151330.40	0.15					151346.20	0.14	150682.50	0.58	150766.30	0.52
		Average			0.30	0.29			0.38		0.27		1.08		0.99
10	500	0.25	118835.77	118565.50	0.24		0.60			118628.60	0.17	116845.40	1.68	117010.90	1.53
		0.50	217503.76	217274.60	0.11		0.27			217326.20	0.08	213561.70	1.81	213935.90	1.64
		0.75	302775.74	302556.00	0.07		0.15			302602.70	0.06	300863.00	0.63	301238.80	0.51
		Average			0.14	0.11	0.34	0.27		0.10			1.37		1.23
30	100	0.25	22305.35	21654.20	2.91							21652.90	2.93	21652.90	2.92
		0.50	41994.84	41431.30	1.34							41402.90	1.41	41404.70	1.41
		0.75	59693.58	59199.10	0.83							59191.70	0.84	59191.70	0.84
		Average			1.70	1.71			1.74				1.73		1.72
30	250	0.25	57554.09	56875.90	1.19					56931.90	1.08	56465.80	1.89	56473.20	1.88
		0.50	107229.81	106673.70	0.53					106711.50	0.48	106105.70	1.05	106132.10	1.02
		0.75	150903.67	150443.50	0.31					150476.90	0.28	150143.10	0.50	150146.30	0.50
		Average			0.68	0.64			0.85		0.61		1.15		1.13
30	500	0.25	116184.37	115473.50	0.61		0.97			115623.70	0.48	114413.80	1.52	114477.20	1.47
		0.50	216729.77	216156.90	0.26		0.43			216274.70	0.21	214526.20	1.02	214646.40	0.96
		0.75	302855.69	302353.40	0.17		0.28			302446.50	0.14	301343.00	0.50	301493.60	0.45
		Average			0.35	0.33	0.56	0.61		0.28			1.01		0.96
Average					0.54	0.53			0.63				1.16		1.10

The proposed model is effective and competitive. This method is easier to implement, has only one parameter to set, and the runtime is much less than other methods. For example, the complete set test of 270 problems was solved in around 34 hours for EO in contrast to the maximum time required for one of the problems for VV-TS which took around 24 hours on a P4 3.2 GHz with 1 GB RAM. For testing whether the results obtained with EO could be improved, a local search based on an exchange of items that are inside the knapsack with those that are outside it, was added. The local search algorithm is described in Algorithm 3. The new EO+LS method improves the results by 5.45% with only a 0.24% rise in the runtime. These results are sufficiently encouraging to further develop this local refinement.

On the whole, EO and EO+LS were able to find near optimal values (around one percent away) for the large test set of problems. Additionally, the results show that as the number of constraints increases, the percentage-gap slowly rises. An important point that must be emphasised is that the value of τ used for the test set of small and large problems was different, $\tau = 1.4$ and $\tau = 2.2$ respectively. This indicates that the value of τ depends on the complexity of the problem to be solved, a point that needs further investigation.

Algorithm 3. LS pseudo-code

```

for each item  $i$  with value of 1 do
  for each item  $j$  with value of 0 do
    if profit of the  $j^{th}$  item > profit of the  $i^{th}$  item then
      if a feasible solution remains when values of the  $j^{th}$  and  $i^{th}$  item are ex-
        changed then
        To do the exchange and update the variables;
      end if
    end if
  end for
end for

```

5 Conclusions

We have introduced an EO model with a novel penalty approach to evaluate the fitness of the items for the MKP. The proposed EO method is an efficient way of finding optimal solutions for small problems and the EO+LS method can obtain competitive solutions for large problems. The strength of the proposed method lies in its low requirements of runtime, memory, parameterisation and implementation compared with other more complex methods. For this reason, the proposed method is suitable for problems that need to obtain rapid results.

The parameter-free penalty approach to evaluate and rank the fitness applied to feasible and infeasible solutions proved to be a competitive mechanism to obtain good results in comparison with other methods. This penalty approach can potentially be applied to other similar constrained problems such as set covering, bin packing and generalised assignment. Investigation of this is our current focus.

Due to an improvement in results when a simple local search was applied, we are also looking to develop appropriate descent local search procedures or develop a hybridisation with simulated annealing, tabu search, genetic algorithms, swarm intelligence, and ant colony optimisation to potentially improve the results of the proposed EO method.

Our results indicate that the value of the parameter τ is sensitive to problem complexity. An interesting challenge could be to work with different τ values, in a computationally efficient way, for different problems, and thereby find better results. For both small and large problems one unique value of τ worked best for each problem.

References

- [1] Boettcher, S., Percus, A.: Evolutionary strategies for extremal optimization: Methods derived from co-evolution. In: Proceedings of GECCO 1999 (1999)
- [2] Bak, P., Tang, C., Wiesenfeld, K.: Self-organized criticality: An explanation of the $1/f$ noise. Physical Review Letters 59(4), 381–384 (1987)

- [3] Bak, P., Sneppen, K.: Punctuated equilibrium and criticality in a simple model of evolution. *Physical Review Letters* 71(24), 4083–4086 (1993)
- [4] Boettcher, S.: Extremal optimization for Sherrington-Kirkpatrick spin glasses. *The European Physical Journal B* 46, 501–505 (2005)
- [5] Boettcher, S., Percus, A.: Extremal optimization: An evolutionary local-search algorithm. In: 8th INFORMS Computer Society Conference (2003)
- [6] Randall, M., Lewis, A.: An extended extremal optimisation model for parallel architectures. In: 2nd International Conference on e-Science and Grid Computing (2006)
- [7] Menaï, M.E., Batouche, M.: An effective heuristic algorithm for the maximum satisfiability problem. *Applied Intelligence* 24(3), 227–239 (2006)
- [8] Randall, M.: Enhancements to extremal optimisation for generalised assignment. In: Randall, M., Abbass, H.A., Wiles, J. (eds.) *ACAL 2007. LNCS*, vol. 4828, pp. 369–380. Springer, Heidelberg (2007)
- [9] Hendtlass, T., Moser, I., Randall, M.: Dynamic problems and nature inspired meta-heuristics. In: 2nd IEEE International Conference on e-Science and Grid Computing (2006)
- [10] Chen, M.R., Lu, Y.Z., Luo, Q.: A novel hybrid algorithm with marriage of particle swarm optimization and extremal optimization. *Optimization Online* (2007)
- [11] Chen, Y.W., Lu, Y.Z., Yang, G.K.: Hybrid evolutionary algorithm with marriage of genetic algorithm and extremal optimization for production scheduling. *International Journal of Advanced Manufacturing Technology* (2007)
- [12] Chu, P., Beasley, J.: A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics* 4(1), 63–86 (1998)
- [13] Gottlieb, J.: Permutation-based evolutionary algorithms for multidimensional knapsack problems. In: 2000 ACM symposium on applied computing, pp. 408–414 (2000)
- [14] Raidl, G.R.: An improved genetic algorithm for the multiconstrained 0-1 knapsack problem (1998)
- [15] Vasquez, M., Hao, J.: A hybrid approach for the 0-1 multidimensional knapsack problem. In: International Joint Conference on Artif. Intell., pp. 328–333 (2001)
- [16] Beasley, J.E.: OR-library: Distributing test problems by electronic mail. *Journal of the Operational Research Society* 41(11), 1069–1072 (1990)
- [17] Bak, P.: *How nature works*. Springer, New York (1996)
- [18] Boettcher, S.: Extremal optimization: Heuristics via co-evolutionary avalanches. *Computing in Science and Engineering* 2, 75–82 (2000)
- [19] Barbosa, H.J.C., Lemonge, A.C.C.: A new adaptive penalty scheme for genetic algorithms. *Information Sciences* 156(3–4), 215–251 (2003)
- [20] Goldberg, D.E.: *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Amsterdam (1989)
- [21] Uyar, S., Eryigit, G.: Improvements to penalty-based evolutionary algorithms for the multi-dimensional knapsack problem using a gene-based adaptive mutation approach. In: *Proceedings of GECCO 2005*, pp. 1257–1264 (2005)
- [22] Vimont, Y., Boussier, S., Vasquez, M.: Reduced costs propagation in an efficient implicit enumeration for the 0-1 multidimensional knapsack problem. *Journal of Combinatorial Optimization* 15(2), 165–178 (2008)

A Generator for Multimodal Test Functions with Multiple Global Optima

Jani Rönkkönen¹, Xiaodong Li², Ville Kyrki¹, and Jouni Lampinen³

¹ Department of Information Technology, Lappeenranta University of Technology,
P.O. Box 20, Lappeenranta, FI-53851, Finland
`jani.ronkkonen@lut.fi`, `ville.kyrki@lut.fi`

² School of Computer Science and IT, RMIT University
`xiaodong@cs.rmit.edu.au`

³ Department of Computer Science, University of Vaasa, P.O. Box 700, Vaasa,
FI-65101, Finland
`jouni.lampinen@uwasa.fi`

Abstract. The topic of multimodal function optimization, where the aim is to locate more than one solution, has attracted a growing interest especially in the evolutionary computing research community. To experimentally evaluate the strengths and weaknesses of multimodal optimization algorithms, it is important to use test functions representing different characteristics and of various levels of difficulty. However, the available selection of multimodal test problems with multiple global optima is rather limited at the moment and no general framework exists. This paper describes our attempt in constructing a test function generator to allow the generation of easily tunable test functions. The aim is to provide a general and easily expandable environment for testing different methods of multimodal optimization. Several function families with different characteristics are included. The generator implements new parameterizable function families for generating desired landscapes and a selection of well known test functions from literature, which can be rotated and stretched. The module can be easily imported to any optimization algorithm implementation compatible with C programming language.

Keywords: Multimodal optimization, test function generator, global optimization.

1 Introduction

Real-world optimization problems often contain multiple global or local optima (i.e. solutions). Multimodal optimization aims to locate all of the global optima of a multimodal function. Evolutionary Algorithms (EAs) have become a popular choice as optimization techniques for many applications and are an interesting candidate for multimodal optimization due to their use of population, which allows multiple solutions to be searched simultaneously. However,

EAs in their original form are typically designed to locate a single global optimum. Many techniques for locating multiple solutions have been developed, commonly referred as *niching* methods [1]. Two most well-known niching methods are probably *crowding* [2] and *fitness sharing* [3]. Apart from *crowding* and *fitness sharing* and their variants, many other niching methods have also been developed [4,5,6,7,8]. It is noticeable, that most of these methods were often evaluated using only 1 or 2 dimensional multimodal test functions, i.e., the functions were not scalable to higher dimensions. Furthermore, these functions are often defined in a specific way, not allowing the functions to be tunable in terms of the characteristics of the multimodal landscapes. For example, for the Shubert function used in [8], as the number of dimensions increases, the number of global optima grows exponentially ($D \cdot 3^D$, where D is the number of dimensions). There is no way to control the number of optima, neither how they are distributed. Additionally, the problem is separable and the global optima are positioned at regular intervals in the search space, both being easily exploitable features. In short, using solely the currently available selection of test functions is typically inadequate for properly analyzing the characteristics of different multimodal optimization algorithms. Especially in the light of no free lunch theorem [9], which states that no optimization algorithm can outperform another over the set of all possible problems, it becomes increasingly important to differentiate the characteristics of the subset of problems each algorithm excels in. For that, a set of parameterizable functions is required, where their characteristics can be changed independently to isolate the effects.

This paper describes an attempt to construct a multimodal test function generator, tailored specially for evaluating the performance of multimodal optimization algorithms in locating multiple globally optimal solutions. Desirable features of such a generator include the following, which have been used as guidelines for designing the generator:

1. Easy to use and tunable.
2. Functions can be transformed from separable to non-separable by rotation.
3. Regular and irregular distributions of optima.
4. Controllable number of global and local optima.
5. Scalable to different dimensions.
6. Reproducible random functions.
7. The software easily expandable and freely available.
8. Facilitates performance measures.

Several function generators have been previously presented in literature: The DF1 [10,11] and Moving Peaks [12] focus on generating dynamic multimodal landscapes, that change over time. Gaviano et.al. [13] generate differentiable multimodal functions by distorting convex functions by polynomials. Constrained test cases generator [14] generates function landscapes by dividing the search space to regions and constructing unimodal function for each, the main feature being the ability to define constraint functions for these regions. The Max set of Gaussians (MSG) landscape generator [15] and the family of generic hump

functions [16] combine several independent peaks to form the function landscape. Among all of the above, only the hump functions have been designed and analyzed in the context of multimodal optimization. The hump family has a limited usability (see section 2.4) and thus a more versatile testing environment is required, which is able to systematically generate a variety of highly tunable, scalable, and controllable multimodal test functions with multiple global optima.

2 The Function Generator

The proposed generator can be used to generate multimodal test functions for minimization, tunable with parameters to allow generation of landscape characteristics that are specifically designed for evaluating multimodal optimization algorithms by their ability to locate multiple optima. At the moment, three families of functions exist, but the generator is easily expandable and new families may be added in future. The current families are cosine, quadratic and common families.

2.1 Cosine Family

The cosine family samples two cosine curves together, of which one defines global minima and another adds local minima. The basic internal structure is regular: all minima are of similar size and shape and located in rows of similar distance from each other. The function family is defined by

$$f_{\cos}(\mathbf{y}) = \frac{\sum_{i=1}^D -\cos((G_i - 1)2\pi y_i) - \alpha \cdot \cos((G_i - 1)2\pi L_i y_i)}{2D} \quad (1)$$

where $y_i \in [0, 1]^D$, the parameters $\mathbf{G}$ and $\mathbf{L}$ are vectors of positive integers which define the number of global and local minima for each dimension and α defines the amplitude of the sampling function (depth of the local minima).

The generator allows the function to be rotated to a random angle and use Bezier curves to stretch each dimension independently to decrease the regularity. To calculate the function value for input vector $\mathbf{x}$, it must be mapped to $\mathbf{y}$. The first step is to calculate $\mathbf{b}$, which is the rotated point corresponding to $\mathbf{x}$:

$$\mathbf{b} = \mathbf{O}\mathbf{x} \quad (2)$$

where the matrix $\mathbf{O} = [\mathbf{o}_1, \dots, \mathbf{o}_D]$ is a randomly generated angle preserving orthogonal linear transformation as described in [17]. The domain of $\mathbf{x}$ (the search space) is the D -dimensional unit hypercube rotated with $\mathbf{O}$. Then $\mathbf{b}$ is mapped to $\mathbf{y}$ by applying the following Bezier formula:

$$y_i = \sum_{j=0}^{n_i} \binom{n_i}{j} P_{i,j} (1 - b_i)^{n_i-j} b_i^j, i = 1, \dots, D \quad (3)$$

where n_i is the degree of the Bezier curve for dimension i and $\mathbf{P}_i$ are the control point vectors for Bezier stretching defined such that $P_{i,0}$ and P_{i,n_i} correspond to the lower and upper bound of y_i and the values between are strictly increasing.

The Bezier stretching will decrease the regularity of the function, but generally not completely eliminate it. The degree of regularity can be roughly estimated by considering the minimum amount of global minima points required to have a set which contains all possible differentials to jump from a neighboring minimum to the next in the axis directions. For completely regular functions, only $D+1$ points are required (as long as there are more than one minimum along each dimension). For stretched functions the required amount is $\sum_{i=1}^D (G_i - 1) + 1$ out of the total amount of global minima $\prod_{i=1}^D G_i$. So the degree of regularity increases along with the dimension. Bezier stretching also affects the shape and size of the minima, increasing the differences in their areas of attraction (AOA).

The amount of minima increases exponentially along dimension and the amount of local minima, which are not global is $\prod_{i=1}^D G_i + (G_i - 1)(L_i - 1)$. For each dimension, two of the outermost minima will always be located on the constraints. This means that if any of the values of $\mathbf{G}$ is less than 3, every minimum will be located on at least one constraint. In the unstretched case, each constraint the minimum sits on, halves the area of attraction (AOA) for that minimum compared to a minimum with one less constraint. The fraction of the AOA from full possible area is thus $1/2^{D-l}$, where $l = 0, 1, \dots, D$ describes the amount of constraints the minimum sits on. $l = 0$ means a minimum located on no constraint (full possible AOA) and $l = D$ is a corner minimum with minimum AOA for dimension D . For methods that rely heavily on the initial points, locating the minima on corners becomes harder with increasing dimensionality, unless the information, that the minima are located on the constraints, is exploited.

Parameter α affects the depth of the local minima. Increasing the value makes the minima deeper, also increasing their area of attraction and thus slightly increasing the difficulty of the problem. Examples of cosine family functions are presented in Figure 1.

2.2 Quadratic Family

The quadratic family can be used to generate completely irregular landscapes. The function is created by combining several minima generated independently. They are described as a D dimensional general quadratic form, where a symmetric matrix $\mathbf{C}$ defines the shape of each minimum. The functions in quadratic family need not be stretched or rotated, because no additional benefit would be gained by doing that to an already irregular function. However, axis-aligned minima may be randomly rotated by rotating the matrix $\mathbf{C}$ as follows:

$$\mathbf{B} = \mathbf{O}\mathbf{C}\mathbf{O}^T \quad (4)$$

The functions are calculated by

$$f_{quad}(\mathbf{x}) = \min_{i=1,2,\dots,q} \left((\mathbf{x} - \mathbf{p}_i)^T \mathbf{B}_i^{-1} (\mathbf{x} - \mathbf{p}_i) + v_i \right) \quad (5)$$

where $x_i \in [0, 1]^D$, $\mathbf{p}_i$ defines the location, and v_i the fitness value of a minimum point for i 'th minimum. q is the amount of minima.

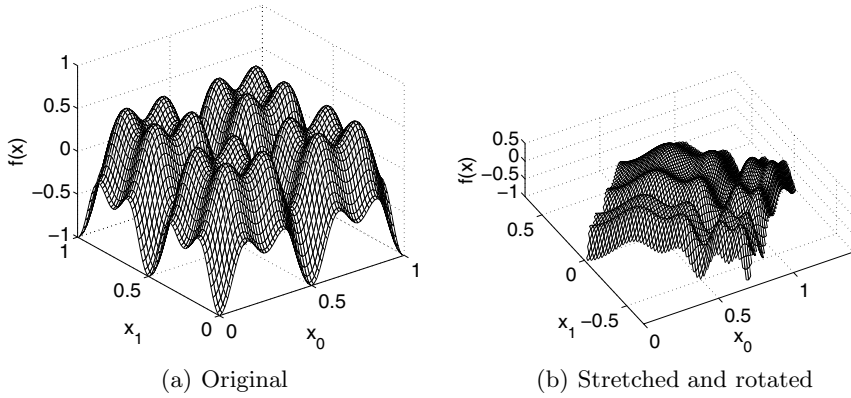


Fig. 1. Example figures of cosine family functions using parameter values $\alpha = 0.8$, $\mathbf{G} = [3, 3]$, $\mathbf{L} = [2, 2]$ (9 global and 16 local minima). Additionally $\mathbf{P}_1 = [0, 0.1, 0.2, 0.5, 1]$ and $\mathbf{P}_2 = [0, 0.5, 0.8, 0.9, 1]$ are used for (b).

Unlike the other function families, the quadratic family allows the number of minima to be defined independently of the dimension and the user may select any number of global and local minima to be generated. The placement of minima is chosen randomly, although the minimum euclidean distance between global minima may be defined. The generator makes sure that no minimum is completely engulfed by another, deeper minimum. The user may also define the minima shape, which may be spherical, ellipsoidal or randomly rotated ellipsoidal. This selection is used for all generated minima. The shape range for global and local minima may be defined independently and the shapes are generated by using uniform random numbers from this range for each dimension in creating matrix $\mathbf{C}$. User may also define the range for the fitness values of local minima points. Figure 2 presents examples of functions from the quadratic family.

Because the locations of minima are random, and the minima will overlap, the sizes of the areas of attraction for different minima will vary greatly. Using fewer or shallower local minima will naturally leave more room for the global minima. Also forcing longer minimum euclidean distance between the global minima will leave more area for each minimum. As the dimensionality increases, the differences in shape parameters will have exponentially increasing effect to the AOAs. Thus the significance of shape range parameters will increase along dimension in deciding the problem difficulty as rather small differences in shape range can lead to large differences to the relative sizes of the AOAs in high dimensions.

To demonstrate this, random start gradient descent (RSGD) algorithm was run on 3 different sets of quadratic functions, with $D = 1, 2, \dots, 10$. The algorithm generates random starting points and descends downhill by performing line searches in numerically estimated gradient direction using bracketing [18, p. 400] and Brent's method [18, p. 404] until all minima have been found. The

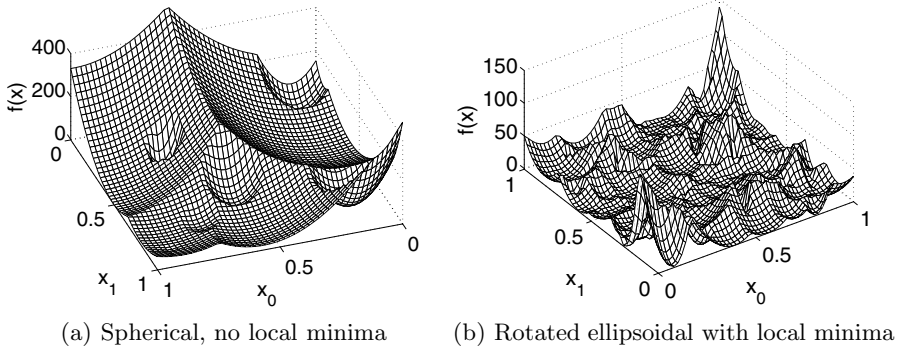


Fig. 2. Examples of quadratic family functions. The minimum euclidean distance between global minima is set to 0.01, and shape range for all minima to $[0.003, 0.03]$. The figure a has 10 global spherical minima, while figure b has 10 global and 100 local rotated ellipsoidal minima such that the local minima points have fitness value in range $[-0.95, -0.15]$. Globally optimal value is always -1.

average performance of RSGD is a decent estimate of the relative sizes of AOAs, because to locate a minimum, the random starting point must be located on the AOA of that particular minimum. If the sizes differ significantly, the larger ones will draw more points, slowing down the process of locating the minima with smaller AOA. Figure 3 displays the results of the runs. The first set of functions has spherical minima with identical shape. The second has also spherical and third an ellipsoidal shape, both allowing the shape range to change by $\pm 50\%$. As can be seen, the required number function evaluations (NFE) as well as the standard deviation increase notably slower along dimension for the first set compared to the other two. This is expected, because the identical shape eliminates the differences on AOA caused by shape. The ellipsoidal shape is hardest on low dimension, but varying spherical shape claims the title on $D > 8$. This is logical, because for spherical shape only one random value is generated, which is then used in all dimensions. If the value is small, it will affect all dimensions. For the ellipsoidal shape, each dimension will get a different random value and the total AOA averages better as more values are generated on higher dimensions. The slower performance on low dimensional ellipsoidal set can be explained by the fact that gradient descent tends to oscillate on non-spherical shapes and thus needs more line searches to find the minimum point compared to spherical shapes, where the gradient points directly to the minimum point.

2.3 Common Family

The common family collects some well known test problems with multiple global optima from literature and implements them inside the generator. The generator allows similar rotation and Bezier stretching, as used in cosine family, for all

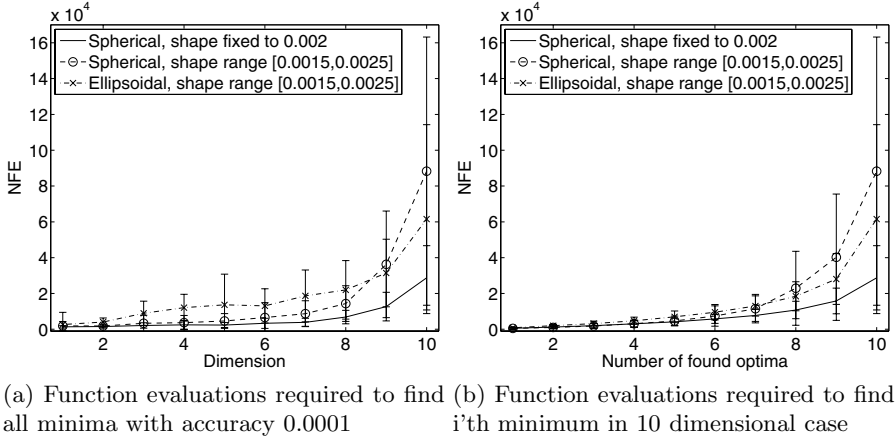


Fig. 3. Performance of RSGD on different quadratic functions with 10 global and no local minima. The minimum euclidean distance between global minima is set to 0.1. The function evaluations are averages from 100 independent runs and figures include standard deviations. For each run, different random seed is used in generating the function.

implemented functions in the common family (the $\mathbf{x}$ is mapped to $\mathbf{y}$ using equations 2 and 3). At the moment, eight different functions are included. One of the functions is a 2 dimensional modification of the well known Rastrigin function [19] to have 4 global and 96 local minima. The function is calculated by formula:

$$f_{ras}(\mathbf{y}) = 20 + \sum_{i=1}^2 (y_i^2 + 10 \cos(2\pi y_i)) \quad (6)$$

where $y_i \in [-5.12, 5.12]^2$. Other implemented two dimensional functions are: Branin [20], Himmelblau [5], Shubert [8], and the Six-hump camel back [21,20]. Additionally Vincent [22] and Deb's 1st and 3rd function [5] have been included with the possibility to scale the dimensionality.

2.4 Comparison to Other Test Function Generators

Singh and Deb recently proposed a scalable multimodal test function family called generic hump functions [16]. Like quadratic family, the hump functions allow irregular landscapes to be generated and the amount of optima to be defined independently of the dimensionality. However, the hump function is defined such that each optimum peak has a fixed radius value and any values outside that are set to 0 and no two peaks can intersect. So the function surface is flat except the peaks. This kind of surface is problematic especially for methods relying on gradient information, because depending on the size and amount of the peaks, large portion of the search space is flat and gradient thus unusable.

However, the problem does not only affect gradient reliant methods, because no method can deduct meaningful direction information from a flat surface. In an extreme case, with very small radius of the peaks, we end up with a needle in a haystack problem, where the vast majority of the search space is flat, only including some very thin peaks. For such a problem, an enumeration technique would be the optimal strategy, because the only viable information available is not to visit the same location twice. In contrast, for the quadratic family, there are no flat areas in the search space and the peaks may freely intersect, making it possible to have different peak inside another peak's area of attraction. Also, for any point, a meaningful numerical estimate for gradient can be calculated. Still, when a reasonably large width for peaks is used, the hump family could provide an interesting environment to test algorithm's ability to handle flat surfaces. For this reason, the hump family would be an interesting addition to the presented generator in future.

The MSG proposed by Gallagher and Yuan [15], is another interesting test function generator. They use a similar approach, as used in quadratic family, to produce peaks independently and use the dominant one to define the function value in point $\mathbf{x}$. However, MSG uses Gaussian density function formula for defining the peaks, which is an exponential function. In comparison, for quadratic family, general quadratic form was selected (which is similar to the exponent part of the Gaussian density function, when the constant is removed) to describe the minima. As a result, the landscapes generated by MSG have much steeper optima shapes compared to the ones generated by quadratic family. In theory, both approaches allow landscapes with no flat areas to be generated. However, a potential problem in using landscapes with steep optima is the numerical accuracy: Because the values ascend towards zero very fast as we move away from an optimum, it is possible that areas are left in the search space where a numerical approximation of gradient rounds to zero, in practice making the area flat from an algorithm's perspective. When using the quadratic form, this is usually not an issue. Of course, the MSG generator is primarily designed for producing landscapes with only one global optimum, while the generator presented here concentrates on cases having multiple global optima, and thus the features included in the generators differ accordingly.

2.5 Implementation and Features

The generator includes an internal constraint handling method to keep the solutions at a given range. The constraints are handled by mirroring the violating value back from the violated boundary by the amount of violation. This makes the function space look continuous for the optimization approach during the run, because any minimum which is located on the boundary looks symmetrical, although in reality the value is calculated in a mirrored point inside the boundaries. The internal constraint handling can be ignored and the linear constraint functions (relevant in rotated cases) can be acquired in analytical form, if required.

Additionally the generator offers a method for deciding how many different globally minimal solutions a given population includes, with a required accuracy. Other useful features included are the possibility of initializing a population with uniform random values in a proper range for the used function, an internal counter for function evaluations and the ability to acquire the amount of minima and the exact locations of global minima. For functions that include random numbers, user may provide a seed for the internal random number generator. With the same seed, the same function is always generated.

The software is written in C programming language obeying ANSI standard and should be easy to import in most optimization algorithms written in C or C++. The generator package is freely available from: <http://www.ronkkonen.com/generator/>, and includes source codes, a simple plotter program written in Matlab for visualizing the 2D functions generated, and detailed documentation.

3 Conclusion and Future Work

In this paper, a function generator module for multimodal problems including several global optima was presented. The generator allows an easy way to construct parameterizable functions and offers a standard environment for testing multimodal optimization algorithms. To the author's knowledge, no versatile generator designed especially for producing problems with multiple global optima exists and the proposed generator aims to mend this deficiency. In addition to offering two novel families of parameterizable functions, the generator implements several well known test problems and allows an easy option to modify them by rotating and stretching. The framework allows an easy addition of new families to the module. We plan to add a variant of the hump family presented by Deb [16] in near future. Also, a variant of the current cosine family, where the global optima would not be located on the constraints, would offer a different challenge for methods that rely on constraints and initial population. For the quadratic family, an option could be added to give the global optima locations as parameters, instead of always generating them randomly.

Acknowledgments

The authors want to thank Harri Lattu and Jarmo Ilonen for their help on implementing the generator software and Jouni Sampo for mathematical consultation.

References

1. Mahfoud, S.: A comparison of parallel and sequential niching methods. In: Proceedings of 6th International Conference on Genetic Algorithms, pp. 136–143 (1995)
2. De Jong, K.: An analysis of the behavior of a class of genetic adaptive systems. PhD thesis, University of Michigan (1975)

3. Goldberg, D., Richardson, J.: Genetic algorithms with sharing for multimodal function optimization. In: Grefenstette, J. (ed.) *Proc. of the Second International Conference on Genetic Algorithms*, pp. 41–49 (1987)
4. Mahfoud, S.: *Niching methods for genetic algorithms*. PhD thesis, Urbana, IL, USA (1995)
5. Beasley, D., Bull, D., Martin, R.: A sequential niche technique for multimodal function optimization. *Evolutionary Computation* 1(2), 101–125 (1993)
6. Harik, G.: Finding multimodal solutions using restricted tournament selection. In: Eshelman, L. (ed.) *Proc. of the Sixth International Conference on Genetic Algorithms*, pp. 24–31. Morgan Kaufmann, San Francisco (1995)
7. Pétrowski, A.: A clearing procedure as a niching method for genetic algorithms. In: *Proc. of the 3rd IEEE International Conference on Evolutionary Computation*, pp. 798–803 (1996)
8. Li, J., Balazs, M., Parks, G., Clarkson, P.: A species conserving genetic algorithm for multimodal function optimization. *Evol. Comput.* 10(3), 207–234 (2002)
9. Wolpert, D., Macready, W., William, G.: No free lunch theorems for search. Technical report, The Santa Fe Institute (1995)
10. Morrison, R., Jong, K.D.: A test problem generator for nonstationary environments. In: *Proceedings of the Congress of Evolutionary Computation*, Piscataway, NJ, pp. 1843–1850. IEEE Press, Los Alamitos (1999)
11. Morrison, R.: *Designing Evolutionary Algorithms for Dynamic Environments*. Springer, Berlin (2004)
12. Branke, J.: *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers, Norwell (2002)
13. Gaviano, M., Kvasov, D., Lera, D., Sergeyev, Y.: Algorithm 829: Software for generation of classes of test functions with known local and global minima for global optimization. *ACM Transactions on Mathematical Software* 29(4), 469–480 (2003)
14. Michalewicz, Z., Deb, K., Schmidt, M., Stidsen, T.: Test-case generator for non-linear continuous parameter optimization techniques. *IEEE Trans. on Evol. Comput.* 4, 197–215 (2000)
15. Gallagher, M., Yuan, B.: A general-purpose tunable landscape generator. *IEEE Transactions on Evolutionary Computation* 10, 590–603 (2006)
16. Singh, G., Deb, K.: Comparison of multi-modal optimization algorithms based on evolutionary algorithms. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, Seattle, WA, pp. 1305–1312. ACM Press, New York (2006)
17. Hansen, N., Ostermeier, A.: Completely derandomized self adaptation in evolution strategies. *Evolutionary Computation* 9(2), 159–195 (2001)
18. Press, W., Flannery, B., Teukolsky, S., Vetterling, W.: *Numerical Recipes in C*, 2nd edn. Cambridge University Press, Cambridge (1992)
19. Törn, A., Žilinskas, A. (eds.): *Global Optimization*. LNCS, vol. 350. Springer, Heidelberg (1989)
20. Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Heidelberg (1996)
21. Ursem, R.: Multinational evolutionary algorithms. In: *Proceedings of Congress of Evolutionary Computation (CEC 1999)*, vol. 3. IEEE Press, Los Alamitos (1999)
22. Shir, O., Bäck, T.: Niche radius adaptation in the cma-es niching algorithm. In: Runarsson, T.P., Beyer, H.-G., Burke, E.K., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) *PPSN 2006*. LNCS, vol. 4193, pp. 142–151. Springer, Heidelberg (2006)

Choosing Leaders for Multi-objective PSO Algorithms Using Differential Evolution

W.R.M.U.K. Wickramasinghe and X. Li

School of Computer Science and Information Technology
RMIT University, Melbourne, VIC 3001, Australia
{uwickram,xiaodong}@cs.rmit.edu.au
<http://www.cs.rmit.edu.au/>

Abstract. The fast convergence of particle swarm algorithms can become a downside in multi-objective optimization problems when there are many local optimal fronts. In such a situation a multi-objective particle swarm algorithm may get stuck to a local Pareto optimal front. In this paper we propose a new approach in selecting leaders for the particles to follow, which in-turn will guide the algorithm towards the Pareto optimal front. The proposed algorithm uses a Differential Evolution operator to create the leaders. These leaders can successfully guide the other particles towards the Pareto optimal front for various types of test problems. This simple yet robust algorithm is effective compared with existing multi-objective particle swarm algorithms.

Keywords: Hybrid, Particle Swarm, Differential Evolution, Multi-objective optimization, Multi-modal problems.

1 Introduction

Particle Swarm Optimization (PSO) has been studied to be an effective Evolutionary Multi-Objective (EMO) algorithm [1]. The main advantage of PSO is the fast convergence [2]. PSO algorithms operate by the notion of *following a leader* to scan the search-space. This movement of following certain particles of a population can become a disadvantage in problems where there are many local optimal fronts. The multi-objective PSO algorithms can prematurely converge onto a local front rather than finding the optimal global front [3, 4]. In our research we developed a mechanism to avoid this downside of multi-objective PSO algorithms. We propose a hybrid PSO algorithm which uses a Differential Evolution (DE) [5] operator to *generate* leaders. This mechanism of generating particles offers the ability to obtain a diverse range of leaders some of which maybe outside a local optimal front. This becomes an advantage to the multi-objective PSO algorithm because now particles are attracted towards these leaders increasing the likelihood of moving the population out of a local optimal front. The proposed hybrid algorithm is simple, robust and efficient in solving a variety of multi-objective problems including problems with many local optimal fronts.

This paper is organized as follows. Section 2 briefly describes the background material including the formal definitions of the PSO and DE variants used in this research. Section 3 presents related work carried out in the field. Section 4 presents our algorithm. The experiments used to evaluate the algorithm will be provided in section 5. Finally, in section 6, we present our conclusions and avenues for future research.

2 Background

We will first present the background material used in this research.

2.1 Particle Swarm Optimization

PSO is a nature-inspired Evolutionary Algorithm (EA) which mimics the behaviour of swarming bees, flocking birds or schooling fish [2]. These behaviours are modelled as rules governing the movement of particles in the search-space. Each particle moves in the search-space by adjusting its position and velocity, which are influenced by its interaction with other particles in its neighbourhood. The i^{th} particle's velocity and position at time t are updated to time $t + 1$ according to the following two equations respectively:

$$\mathbf{v}_i(t + 1) = \chi(\mathbf{v}_i(t) + \phi_1(\mathbf{p}_i - \mathbf{x}_i(t)) + \phi_2(\mathbf{p}_g - \mathbf{x}_i(t))) \quad (1)$$

$$\mathbf{x}_i(t + 1) = \mathbf{x}_i(t) + \mathbf{v}_i(t) \quad (2)$$

This version of PSO is known as the *Constriction Type 1* as defined in [6]. Here, $\mathbf{v}_i$ is the velocity of the i^{th} particle and $\mathbf{x}_i$ is its position. The variables ϕ_1 and ϕ_2 are random numbers generated uniformly between $[0, \frac{\varphi}{2}]$. Here, φ is a constant equal to 4.1 [6]. $\mathbf{p}_i$ is the best position found by the particle (also known as *personal best*); $\mathbf{p}_g$ is the best position found in the particle's neighbourhood (also known as *global best* or *leader*). χ is called the constriction factor, and is used to prevent a particle from exploring too far in the search-space. We used $\chi = 0.7298$, which was calculated from $\frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|}$ [6]. Essentially each particle moves towards somewhere between its personal best and the global best.

2.2 Differential Evolution

DE differs from other EAs in their mechanism of generating offspring. In EAs, an individual plays the role of a parent to generate an offspring. This is similar in PSO where a particle updates its velocity to move to another position, which mimics the creation of an offspring, if the original particle was considered as a parent. In DE, an offspring is generated by using vector differences among individuals in the population [5].

There are many schemes of generating individuals in DE. Here, we present the simplest and most popular scheme, the **DE/rand/1/bin**, which is used in this

research. For an individual vector $\mathbf{x}_i$ a trial vector $\mathbf{u}_i$ is generated using three other individuals $\mathbf{x}_{r1}, \mathbf{x}_{r2}, \mathbf{x}_{r3}$ from the population such that $i \neq r1 \neq r2 \neq r3$. The decision variable j of an individual is generated using equation (3).

$$\mathbf{u}_i = u_{j,i} = \begin{cases} x_{j,r1} + F(x_{j,r2} - x_{j,r3}) & \text{if } (rand_j < CR \text{ or } j = j_{rand}) \\ x_{j,i} & \text{otherwise} \end{cases} \quad (3)$$

here, $j_{rand} \in [1, D]$ where D is the number of dimensions in the search-space. $F \in [0, 1]$ is called the scaling factor. CR is the crossover ratio and $rand_j$ is randomly generated uniformly between $[0, 1]$. The traditional values used for DE/rand/1/bin are $CR = 0.9$ and $F = 0.5$.

In a traditional multi-objective DE algorithm if $\mathbf{u}_i$ dominates $\mathbf{x}_i$ then $\mathbf{x}_i$ is replaced by $\mathbf{u}_i$, if not $\mathbf{u}_i$ is discarded [7]. In the proposed algorithm the trial vector $\mathbf{u}_i$ is chosen as the leader ($\mathbf{p}_g$) for the particle $\mathbf{x}_i$. The particle will update its velocity and positions according to equations (1) and (2) using this $\mathbf{p}_g$.

We observed that the traditional values for CR and F are unsuitable for multi-objective optimization problems from our initial experiments. A higher CR value allows less frequent crossover operations. This, contributes in placing leaders far from the vicinity of other particles in the search-space. These leaders could be placed in worst positions than the current particles. To overcome this issue we used $CR = 0.2$ and $F = 0.4$, which locates the leaders near the vicinity of other particles. A lower CR value is also present as desirable in other examples of multi-objective optimization algorithms using DE in the literature [7, 8, 9, 10].

3 Related Work

There are several examples of multi-objective PSO algorithms where leaders are chosen by the notion of *dominance* in the literature. In DNPSO [11], particles choose a leader from their neighbours. Here, a particle will have a set of dynamically changing neighbours and it will choose the best particle among them as a leader. NSPSO [3] adopts the non-dominated sorting mechanism proposed in NSGA-II [12]. In NSPSO, leaders are chosen from the first non-dominated front of the current population. Here, each particle chooses a leader randomly from the top 10% of least crowded particles of the first non-dominated front. In a later study maximinPSO [4] was proposed, which used the maximin fitness function [13] to derive dominance. For every iteration a set of particles was extracted using the maximin fitness. From this set the top 10% of particles with the least maximin value was chosen as candidates to be leaders. Each particle will create a leader by selecting values from all dimensions from this set of particles. In maximinPSO, a particle's leader is not present in the population, but created from several candidates for leaders. The difference from the approach of creating leaders in maximinPSO and the proposed algorithm is that leaders can be positioned outside the first non-dominated front.

In other studies of multi-objective PSO algorithms, leaders were selected from an external archive. An elite archive is used in [14] to store the non-dominated individuals from every iteration. The archive has a special data structure called

a *dominance tree* which is updated with the non-dominated particles found in each run. The algorithms proposed in [15] and [16] both use an archive to store particles which are candidates for leaders. In [15], the non-dominant particles are added to the archive in each iteration. This archive is truncated to a fixed size according to the crowding distance of the particles. This ensures that the least crowded particles will be candidates for leaders. The *Sigma method* is used in [16] to obtain suitable particles as leaders in each iteration. These leaders are then stored in the archive in each iteration and truncated so that the particles with smallest sigma values are retained. Recently, in [17] a methodology of choosing a random particle from the population as a leader was proposed. A particle will adjust its velocity and position and then according to some replacement rules it will decide whether to replace this leader or another particle if it is in a dominant position. In all of these multi-objective PSO algorithms the leaders are extracted by members of the population. In contrast, in our proposed hybrid algorithm leaders are not necessarily particles of the current population.

It is useful to notice that there have been several hybrid single-objective DE and PSO algorithms proposed in the literature. A study in [18] proposes a PSO algorithm, where from time to time the DE operator is used to create an individual which will replace a particle than using the PSO rules to move it. DEPSO [19] uses the DE operator to updating the personal best position of particles. However, this updating procedure is done once every couple of iterations. A similar approach to [18] was proposed in [20] where the particles' movement is determined partly by a DE operator and partly by PSO update rules. In a recent study [21] the PSO update rules were modified by using a DE operator. Here, every particle underwent this modified PSO rules to move in the search-space. In all these algorithms the movement of the particle was influenced by a DE operator. In our proposed hybrid algorithm the particles move following the PSO rules, while only a leader is created using a DE operator. This mechanism ensures a faster convergence because of the PSO update rules, while retaining a multi-objective PSO's ability to move outside a local optimal front because of a diverse range of leaders generated by DE.

4 The MDEPSO Algorithm

The proposed MDEPSO algorithm has the following steps:

- **Step 1: Initialize the particles**

A population of size N is first initialized. Here, a particle's decision variables are initialized from equation (4).

$$rand(0.0, 1.0) * (UB - LB) + LB \quad (4)$$

here, $rand(0.0, 1.0)$ represents a random number generated uniformly between $[0.0, 1.0]$. LB and UB are the lower-bounds and upper-bounds respectively of the decision variables of a multi-objective problem instance. The velocity is initialized to a random value in the interval $[0, UB - LB]$. The

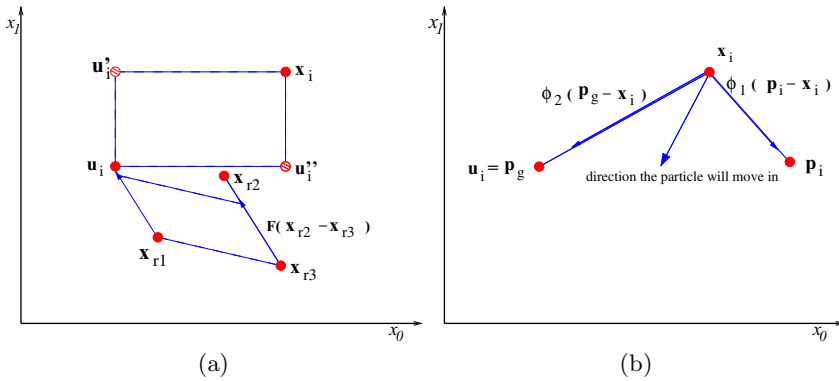


Fig. 1. (a) Creating a leader vector using the DE operator (b) Applying the PSO rules to move the particle

personal best of an individual is set to the current values of the decision variables. Half of the populations' direction is reversed by setting the velocity to negative according to a coin toss to better explore the search-space. The particles are then evaluated with the objective functions and fitness is assigned. The fitness values are used to determine the dominance.

– **Step 2: Generate leaders and move the particles**

Each particle $\mathbf{x}_i$ will choose three other particles $\mathbf{x}_{r1}, \mathbf{x}_{r2}, \mathbf{x}_{r3}$ from the population such that $i \neq r1 \neq r2 \neq r3$. Then a leader vector ($\mathbf{u}_i = \mathbf{p}_g$) is derived from the DE operator in equation (3). Each particle will move towards their generated leader vectors (figure 1) updating their velocities and positions according to the PSO rules in equations (1) and (2).

– **Step 3: Update the particles' personal bests**

The particles are evaluated according to the objective functions and fitness is assigned. Next, the particles' personal bests are updated according to their current positions and best positions found so far.

– **Step 4: Obtain the particles to move to the next iteration**

The population of N particles at the beginning of the iteration is combined with the N number of particles that have changed their positions to create a population of size $2N$. The non-dominated sorting process [12] is applied to this $2N$ population to obtain N particles which are carried over to the next iteration.

The steps 2 to 4 are repeated until the maximum number of iterations is reached.

Advantage of Generating Leaders Using DE

Figure 2 shows the generated leaders for the two-objective ZDT4 [22] test problem within an iteration. The DE operator takes the differences between vectors

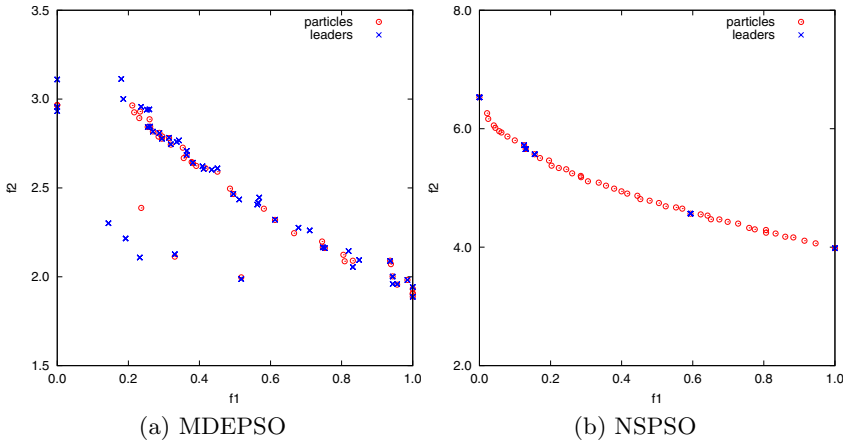


Fig. 2. Positions of leaders for ZDT4 within an iteration for MDEPSO and NSPSO

to generate a leader. This becomes an advantage because there is a chance for particles not in the first non-dominated front to be selected in the process of creating a leader. The vector differences between a particle in the first non-dominated front and one in a different front can be large, that the generated leader will be in a position outside the first non-dominated front. These leaders can now successfully attract other particles to move from the current local optimal front towards the global optimal front. In other multi-objective PSO algorithms particles would rarely move outside a local optimal front because all leaders are trapped to the same front. This feature of generating leaders outside of the current local front, gives MDEPSO the ability to escape a local front.

Selecting leaders from the first non-dominated front can also be a disadvantage for multi-objective PSO algorithms mainly because many particles could follow the same leader. This restrains the PSO algorithm from searching the search-space effectively.

5 Experiments

We used the following problems; two-objective ZDT [22], three-objective DTLZ [23], and two-objective WFG [24] test suites. These test problem suites contain many varieties of multi-objective problems including multi-modal problems (many local optimal fronts with one global optimal front). We present the Normalized Hyper-Volume metric [25], which provides the convergence and spread of solutions along the solution front for comparing the effectiveness of MDEPSO against NSPSO [3], maximinPSO [4], OMOPSO [15] and NSGA-II [12] on the same test problems.

For each problem a constant population of 200 individuals were used. All the algorithms were executed for a maximum of 750 iterations on all test problems. For simpler problems like ZDT1-ZDT3, MDEPSO found the Pareto optimal

Table 1. Normalized Hyper-Volume for MDEPSO, NSPSO, maximinPSO, OMOPSO and NSGA-II on ZDT, DTLZ and WFG test problem suites

	MDEPSO	NSPSO	maximinPSO	OMOPSO	NSGA-II
ZDT4	<i>0.0000</i> ± 0.0000	<i>0.0000</i> ± 0.0000	<i>0.0000</i> ± 0.0000	0.0022 ± 0.0114	0.6636 ± 0.0001
ZDT6	0.4033 ± 0.0001	0.4033 ± 0.0001	0.4027 ± 0.0002	0.4014 ± 0.0001	0.4025 ± 0.0002
WFG3	0.4428 ± 0.00001	0.4426 ± 0.0001	0.4413 ± 0.0006	0.4422 ± 0.0000	0.4428 ± 0.0001
WFG4	0.2175 ± 0.0014	0.2161 ± 0.0008	0.2163 ± 0.0016	0.2151 ± 0.0019	0.2205 ± 0.0001
WFG5	0.1995 ± 0.0035	0.1974 ± 0.0002	0.1975 ± 0.0001	0.1963 ± 0.0001	0.1973 ± 0.0001
DTLZ1	0.7749 ± 0.0001	<i>0.0000</i> ± 0.0000	<i>0.0000</i> ± 0.0000	<i>0.0000</i> ± 0.0000	0.7852 ± 0.0018
DTLZ3	<i>0.0000</i> ± 0.0000	<i>0.0000</i> ± 0.0000	<i>0.0000</i> ± 0.0000	<i>0.0000</i> ± 0.0000	0.4084 ± 0.0029
DTLZ6	0.0959 ± 0.0000	0.0909 ± 0.0214	0.0956 ± 0.0001	0.0949 ± 0.0000	0.0957 ± 0.0000
DTLZ7	0.3094 ± 0.0013	0.1818 ± 0.1381	0.2827 ± 0.0553	0.2758 ± 0.0030	0.3045 ± 0.0013

fronts in under 50 iteration, but for consistency the maximum number of iterations were fixed. Each algorithm was executed for 50 runs on each test problem and the results were averaged. The parameter values for MDEPSO, NSPSO and maximinPSO were the standard values presented in Constriction Type 1" PSO. $CR = 0.2$ and $F = 0.4$ was used for the DE operator in MDEPSO. An archive size of 100 and mutation probability of 0.5 was used in OMOPSO. In NSGA-II, a crossover probability of 0.9 on a SBX crossover operator was used. The mutation probability was set as $1/D$, where D is the number of dimensions of the problem instance.

Table 1 shows the normalized hyper-volume metric of all the algorithms on the test problems used in this study. Here, we have only illustrated some of the test problems due to the limitation of space. MDEPSO on average outperforms all the other multi-objective PSO algorithms. However, for ZDT4 and DTLZ3, MDEPSO was unable to converge to the global Pareto front. We introduced a mutation mechanism for MDEPSO to overcome this.

The MDEPSO algorithm's Step 3 is updated with a mutation operator. A particle is mutated with a given probability using a PSO mutation rule. Choose a particle for mutation if probability is less than some δ . A particle chosen for mutation will update each decision variable j if the probability is less than δ . The decision variable is *mutated* according to equation (5). We used $\delta = 0.1$ as the mutation probability. Each particle will update its personal best after the mutation is applied.

$$x_{j,i} = (x_{j,i} + rand(LB, UB))/2 \quad (5)$$

Table 2. Normalized Hyper-Volume for MDEPSO, NSPSO and maximinPSO after the mutation operator was added compared with OMOPSO and NSGA-II on ZDT4, WFG4, DTLZ1 and DTLZ3

	MDEPSO	NSPSO	maximinPSO	OMOPSO	NSGA-II
ZDT4	0.6628 $\pm$ 0.0005	0.1331 $\pm$ 0.0053	0.2448 $\pm$ 0.0514	0.0022 $\pm$ 0.0114	0.6636 $\pm$ 0.0001
WFG4	0.2196 $\pm$ 0.0005	0.2187 $\pm$ 0.0008	0.2185 $\pm$ 0.0015	0.2151 $\pm$ 0.0019	0.2205 $\pm$ 0.0001
DTLZ1	0.7754 $\pm$ 0.0001	0.0840 $\pm$ 0.0059	0.6925 $\pm$ 0.0008	0.0000 $\pm$ 0.0000	0.7852 $\pm$ 0.0018
DTLZ3	0.3628 $\pm$ 0.0053	0.0000 $\pm$ 0.0000	0.0586 $\pm$ 0.0481	0.0000 $\pm$ 0.0000	0.4084 $\pm$ 0.0029

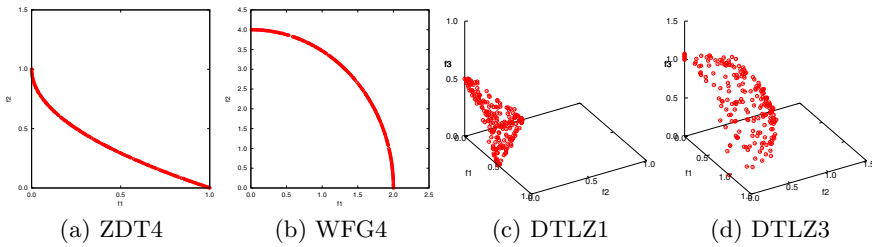


Fig. 3. Solution fronts obtained by MDEPSO on ZDT4, WFG4, DTLZ1 and DTLZ3

We employed this same mutation rule on NSPSO and maximinPSO also. OMOPSO already has a mutation rule in its functionality. After 50 runs of the modified MDEPSO we observed that for all of the test problems MDEPSO was able to locate the global optimal front. Table 2 shows the normalized hyper-volume values after the mutation operator was added. We have listed only values for the multi-modal problems because for the other problems the normalized hyper-volume values were similar.

Comparing the results from Table 1 and 2 it is clear that MDEPSO outperforms the other multi-objective PSO algorithms in the test problems, especially for the multi-modal problems. One of the main objectives of this research was to find a method for multi-objective PSO algorithms to better handle problems with many local fronts. This was possible with the selection method of leaders in MDEPSO with mutation. On average MDEPSO gives a higher hyper-volume value, which is slightly better than NSGA-II in locating the Pareto optimal fronts on a range of different test problems. However, the crux of the research was not to outperform NSGA-II. We have included the results of NSGA-II to compare the performance between the other PSO algorithms and MDEPSO. MDEPSO is capable of locating the global Pareto front of all the test problems where other PSO algorithms were unable to.

Figure 3 shows the final non-dominated solution sets obtained by MDEPSO with mutation at the end of 750 iterations for the multi-modal problems.

6 Conclusion and Future Work

The proposed hybrid MDEPSO algorithm is effective in locating the Pareto optimal front in difficult multi-modal multi-objective problems than existing PSO algorithms. The unique feature of generating and selecting leaders provides a greater range of directions and positions particles can move to, including those positions outside a local optimal front. MDEPSO is a simple and robust EMO algorithm which has a comparable performance to NSGA-II.

It will be interesting to study if other combination mechanisms can be used to generate leaders other than DE. We also would like to investigate if this hybrid algorithm is capable of solving rotated multi-objective problems, where many state-of-art EMO algorithms are unsuccessful.

References

1. Reyes-Sierra, M., Coello Coello, C.A.: Multi-Objective Particle Swarm Optimizers: A Survey of the State-of-the-Art. *International Journal of Computational Intelligence Research* 2, 287–308 (2006)
2. Kennedy, J., Eberhart, R.C.: *Swarm intelligence*. Morgan Kaufmann Publishers Inc., San Francisco (2001)
3. Li, X.: A non-dominated sorting particle swarm optimizer for multiobjective optimization. In: Cantú-Paz, E., Foster, J.A., Deb, K., Davis, L., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Kendall, G., Wilson, S.W., Harman, M., Wegener, J., Dasgupta, D., Potter, M.A., Schultz, A., Dowsland, K.A., Jonoska, N., Miller, J., Standish, R.K. (eds.) *GECCO 2003*. LNCS, vol. 2723, pp. 37–48. Springer, Heidelberg (2003)
4. Li, X.: Better spread and convergence: Particle swarm multiobjective optimization using the maximin fitness function. In: Deb, K., et al. (eds.) *GECCO 2004*. LNCS, vol. 3102, pp. 117–128. Springer, Heidelberg (2004)
5. Price, K., Storn, R.M., Lampinen, J.A.: *Differential Evolution: A Practical Approach to Global Optimization*. Natural Computing Series. Springer, Secaucus (2005)
6. Clerc, M., Kennedy, J.: The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation* 6, 58–73 (2002)
7. Abbass, H., Sarker, R., Newton, C.: PDE: a pareto-frontier differential evolution approach for multi-objective optimization problems. *IEEE Congress on Evolutionary Computation (CEC)* 2, 971–978 (2001)
8. Huang, V.L., Suganthan, P.N., Qin, A.K., Baskar, S.: Multiobjective differential evolution with external archive and harmonic distance-based diversity measure. In: *School of Electrical and Electronic Engineering Nanyang, Technological University Technical Report* (2005)
9. Kukkonen, S., Lampinen, J.: GDE3: the third evolution step of generalized differential evolution. In: *IEEE Congress on Evolutionary Computation (CEC)*, pp. 443–450 (2005)
10. Robic, T., Filipic, B.: Demo: Differential evolution for multiobjective optimization. In: Coello Coello, C.A., Hernández Aguirre, A., Zitzler, E. (eds.) *EMO 2005*. LNCS, vol. 3410, pp. 520–533. Springer, Heidelberg (2005)

11. Hu, X., Eberhart, R., Shi, Y.: Particle swarm with extended memory for multiobjective optimization. In: IEEE Swarm Intelligence Symposium (SIS), pp. 193–197 (2003)
12. Deb, K., Agrawal, S., Pratab, A., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 182–197 (2002)
13. Balling, R.: The maximin fitness function; multi-objective city and regional planning. In: Fonseca, C.M., Fleming, P.J., Zitzler, E., Deb, K., Thiele, L. (eds.) EMO 2003. LNCS, vol. 2632, pp. 1–15. Springer, Heidelberg (2003)
14. Fieldsend, J., Everson, R., Singh, S.: Using unconstrained elite archives for multiobjective optimization. *IEEE Transactions on Evolutionary Computation* 7, 305–323 (2003)
15. Reyes-Sierra, M., Coello Coello, C.A.: Improving pso-based multi-objective optimization using crowding, mutation and epsilon-dominance. In: Coello Coello, C.A., Hernández Aguirre, A., Zitzler, E. (eds.) EMO 2005. LNCS, vol. 3410, pp. 505–519. Springer, Heidelberg (2005)
16. Mostaghim, S., Teich, J.: Strategies for finding good local guides in multi-objective particle swarm optimization. In: IEEE Swarm Intelligence Symposium (SIS), pp. 26–33 (2003)
17. Allmendinger, R.: Reference point-based particle swarm optimization using a steady-state approach: Master's thesis (2008)
18. Hendtlass, T.: A combined swarm differential evolution algorithm for optimization problems. In: Monostori, L., Váncza, J., Ali, M. (eds.) IEA/AIE 2001. LNCS, vol. 2070, pp. 11–18. Springer, Heidelberg (2001)
19. Zhang, W.J., Xie, X.F.: DEPSO: Hybrid particle swarm with differential evolution operator. In: IEEE International Conference on Machine Learning and Cybernetics (ICMLC), pp. 3816–3821 (2003)
20. Hao, Z.F., Guo, G.H., Huang, H.: A particle swarm optimization algorithm with differential evolution. *IEEE International Conference on Machine Learning and Cybernetics (ICMLC)* 2, 1031–1035 (2007)
21. Xu, X., Li, Y., Fang, S., Wu, Y., Wang, F.: A novel differential evolution scheme combined with particle swarm intelligence. In: IEEE Congress on Evolutionary Computation, CEC (2008)
22. Zitzler, E., Deb, K., Thiele, L.: Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation* 8, 173–195 (2000)
23. Deb, K., Thiele, L., Laumanns, M., Zitzler, E.: Scalable test problems for evolutionary multi-objective optimization. In: *Evolutionary Multiobjective Optimization (EMO): Theoretical Advances and Applications*, pp. 105–145. Springer, Heidelberg (2005)
24. Huband, S., Hingston, P., Barone, L., While, R.L.: A review of multiobjective test problems and a scalable test problem toolkit. *IEEE Transactions on Evolutionary Computation* 10, 477–506 (2006)
25. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation* 3, 257–271 (1999)

Comparison between Genetic Algorithm and Genetic Programming Performance for Photomosaic Generation

Shahrul Badariah Mat Sah¹, Vic Ciesielski¹, Daryl D'Souza¹,
and Marsha Berry²

¹ School of Computer Science and Information Technology

² School of Creative Media,

RMIT University, GPO Box 2476V Melbourne Victoria 3001, Australia

smatsah@cs.rmit.edu.au,

{vic.ciesielski, daryl.dsouza, marsha.berry}@rmit.edu.au

Abstract. Photomosaics are a new form of art in which smaller digital images (known as tiles) are used to construct larger images. Photomosaic generation not only creates interest in the digital arts area but has also attracted interest in the area of evolutionary computing. The photomosaic generation process may be viewed as an arrangement optimisation problem, for a given set of tiles and suitable target to be solved using evolutionary computing. In this paper we assess two methods used to represent photomosaics, genetic algorithms (GAs) and genetic programming (GP), in terms of their flexibility and efficiency. Our results show that although both approaches sometimes use the same computational effort, GP is capable of generating finer photomosaics in fewer generations. In conclusion, we found that the GP representation is richer than the GA representation and offers additional flexibility for future photomosaics generation.

Keywords: Photomosaic, Genetic Programming (GP), Genetic Algorithm (GA).

1 Introduction

Photomosaics are a new form of digital mosaic composed of a tessellation of thumbnail pictures known as tiles. When viewed from afar, the subject becomes evident as we perceive the association of the tiles rather than the individual tiles. When viewed close up, the subject is invisible as the details of each tile emerge. An example of a photomosaic appears in Figure 1.

In a common approach photomosaics are generated by distributing one or more copies of tiles from among a set of small, carefully selected image tiles, across a two-dimensional gridded canvas. Besides being artistically interesting, a photomosaic may be viewed as a solution to a combinatorial optimisation problem based on the examples given by Mitchell et al [3]. Since photomosaic generation involves a tile collection and a set of fixed locations on a gridded

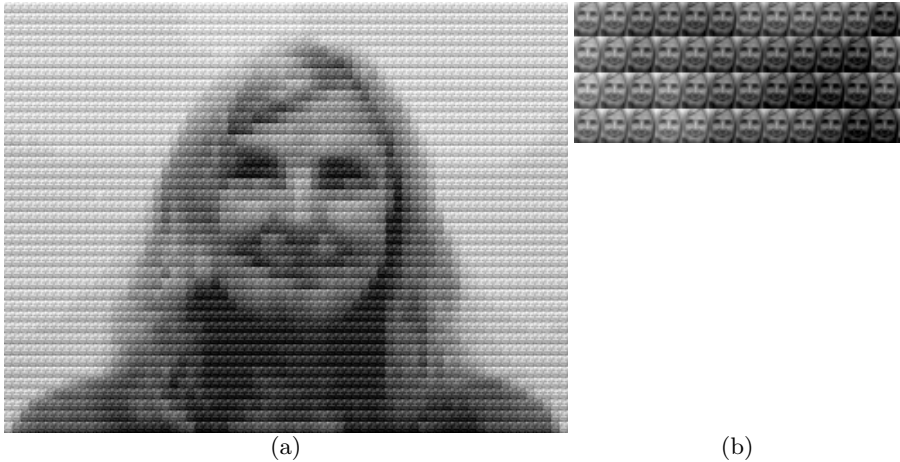


Fig. 1. Example of (a) a photomosaic and (b) a close-up of the tiles

canvas, it may be likened to the problem of attaining an optimal distribution of limited available resources, a problem well-suited to resolution via evolutionary computing. The use of evolutionary computing also allows parallel search for many possible solutions in a single run. Photomosaics are of interest to both artists and computer scientists, and indeed, our work in photomosaic generation has involved ongoing collaboration with artists. From our experiences of working with artists, two important aims of photomosaic generation have been established. First it is necessary to be able to arrive at the final photomosaic as soon as possible, that is, with the smallest number of generations. Secondly, it is desirable to have a flexible development environment, in order to be able to easily implement new creative suggestions from the artists, for example, a new tile arrangement strategy. We anticipated that genetic algorithms (GAs) will be more suited to the first goal, as we expect that GAs use fewer evaluations to generate better photomosaics, while genetic programming (GP) will be better suited to the second goal. As such, the aim of this paper is to compare the GA and GP representations of photomosaic problem. A more generic treatment of GA and GP problem representations appear, respectively, in [3] and [2]. Furthermore, some experiments were carried out to seek answers for improved representations, in terms of the following criteria.

- Number of generations needed to generate the best photomosaic.
- Development flexibility for artistic variation implementation.

The remainder of this paper is structured as follows. Section 2 discusses previous research related to photomosaic generation and to comparative studies between GA and GP. In Section 3 we describe the overall strategy to generate a photomosaic and present our formula to determine the fitness of a solution. Section 4 provides the experimental backdrop to the assessment of the differences between our GA and GP implementations for photomosaic generation.

The results of these experiments are presented in Section 5. Finally, Section 6 summarises our findings and provides pointers to future research direction in the area of photomosaic generation.

2 Related Work

Research in computer-generated photomosaics was pioneered by Silvers [5] in 1997. Silvers' approach relied on a large image repository in order to maximise the probability of finding matches between a given target image and a set of tiles. Silvers' pioneering work attracted further interest in photomosaic generation, and ranging from performance improvements to variations in problem-solving strategy. We briefly review the most relevant of such work.

Finkelstein and Range [6] applied wavelets-based image matching between potential tiles and target image in order to speed up the generation process. Di Blasi et al. focused on technique to speed up the tile search process by using Antipole Tree structure [7]. Images from a database were first classified using the tree structure before any photomosaic was constructed.

Other work has focused on creating forms of photomosaics using different tile placement strategy. The Jigsaw Image Mosaic (JIM), created by Kim and Pelacini [8], is able to create arbitrary-shaped mosaics using arbitrary shaped tiles. The use of stacking layers of tiles for photomosaic generation was introduced by Park [9]. However, in both of these studies the use of exhaustive search was employed, to generate photomosaics.

More recently work in photomosaics has explored the use of evolutionary algorithms in the generation process. Ciesielski et al. [4] investigated the use of GA to produce frames for animated photomosaics. Small sets of generic images and miniature portraits have been used in the photomosaic generation. The work of Wijesinghe et al. [10] explored the use of random tile placement using genetic programming GP. Working with small set of tiles, this study allowed rotation and overlapping between tiles to create the photomosaics.

In this paper we present results of a comparative study of GA and GP for photomosaic generation. Comparative studies between evolutionary algorithms is not a new idea, with several having been completed for different problems to better understand the use of evolutionary algorithms in arriving at good solutions. In a comparative study by Sinclair and Shami [12], in the area of software agents, the comparison is made using almost similar problem formulations, with one function discarded from the GP approach for the ease of implementation and some modifications to the range of arguments in the GP program. Meanwhile the GA implementation is a simplified version of an existing GA-based work of Maskell and Wilby(mentioned in [12]). Although the study served as preliminary work for a larger project to evolve complex software agents for telecommunication network, the results of the study provided some insight into performance and its dependency on having a focused set of functions for particular problems, regardless of the GA or GP implementation. However, the concluding decision to proceed with GP implementation was based on the fewer number of generations

from the experiments. Another comparative study between GA and GP, in the area of robot control [13], it was discovered that using different approaches, the problem of finding the best motions for a robot to score a goal required different representations and there are some trade-offs in each implementation. Based on these previous studies, it has been shown that the implementation of a problem in GA and GP is not directly comparable. Hence, it is of interest to us to assess the performance of GA and GP to find a better representation for photomosaic generation.

3 Problem Formulation

To generate a photomosaic in each of the GA and GP approaches, we start with a blank canvas, the size of the target image. The canvas is logically viewed as comprising a grid of width K pixels and height of L pixels. For tiles each of width k pixels and height l pixels, the canvas will accommodate $K/k = n$ tiles in each row, and $L/l = m$ tiles in each column. Tiles to be placed on the canvas are randomly selected from a tile collection. The size of each tile in this collection is set sufficiently small so as to ensure that, together with other tiles of the same size, it forms a clear photomosaic, whilst at the same time large enough to show the tile details when viewed close up. The total number of tiles for the given target is $m \times n = C$. The range of integers $[0, C - 1]$ is used to index the cell locations on the gridded canvas. As such, cells are filled in sequential order starting from the top left corner to bottom right corner of the grid, row by row.

Fig 2 provides a simple example of the use of the canvas space. The canvas in the example can accommodate a total of 6 tiles, that is, $n = 2$ (the number of tiles in each column), and $m = 3$ (the number of tiles in each row). The tile positions are numbered 0 to 5, and taken row by row. In our implementation a tile can be reused, as in the case of cell locations 0 and 2. In order to fill the cells, tile selection is made from tile collection, implemented as a list. The tile selection is done randomly using the indices of the tile collection. In the example provided, tiles numbered 15 and 10 represent the 15th and the 10th tiles in the tile collection. In this study, we are more interested in computational analysis than the artistic output. Therefore, a small-sized target image and a small tile collection (i.e. 16 flat-shaded potential tiles) were chosen. The size of the canvas and the target image were set at 120×100 pixels and the size of each tile was set at 5×5 pixels. The grid can fit $n = 24$ tiles in each row and $m = 20$ tiles in each column.

3.1 Fitness Evaluation

The fitness of an individual (photomosaic) is calculated as the *sum of pixels differences* between the candidate photomosaic and the target image with i and j referring to a pixel's position, and is given by the following formulation.

$$\sum_{i=1}^K \sum_{j=1}^L |target(i,j) - individual(i,j)|. \quad (1)$$

Here, $target(i,j)$ is the pixel value of i^{th} location on K width and j^{th} location on L height of the target image, and likewise for $individual(i,j)$ on a generated photomosaic. This measure is further *normalised* to scale to the range of $]0, 1]$ by dividing the final value with the total number of pixels on the target.

4 Experiments: GA and GP Approaches

We present our implementations for the GA and GP to generate photomosaics, in the following subsections. The common parameters of evolutionary algorithms such as population size and number of generations are defined to be equal. However, the GA and GP programs use different mutation rate, suitable for each approach. Details of parameters for both GA and GP implementations are listed in Table 1. To support our discussion Fig 2 presents a simplified example of a target image, an evolved photomosaic and a table of selected tiles. The table also contains the position of each selected tile on the grid. Additional parameters for each implementation are explained in the designated subsections.

4.1 GA-Based Program

The GA chromosome is a concatenation of C tiles in a single list, which in this case is 480 tiles. A chromosome is generated by placing the first selected tile from the tile collection in the first cell of the chromosome, the second selected tile is placed in the second cell of the chromosome, and so on. The sequence of selected tiles in the chromosome correlates to the tile positions on the grid. As the grid is filled sequentially, the first tile from the chromosome be placed in the first cell of the gridded canvas, the second selected tile is placed in the second cell of the gridded canvas, and so on.

A chromosome of the initial population is generated by randomly selecting tiles from the tile collection, and placing the tiles, as they are selected, onto the canvas as described above. We implemented the GA program using SGA-C [11]. GA parameters used are presented in Table 1. In addition, a maximum number of 9 mutations were allowed for a single chromosome. Based on the fitness value, a new best photomosaic is identified from each evolutionary run and written out (saved) as an image file. Fig 2 shows an example of a GA chromosome for the given target image.

4.2 GP-Based Program

To represent the photomosaic problem as a GP parse tree, we provide one function and one terminal as the building blocks. The function *TileJoint* allows up to three terminals to be linked together in the tree structure and does not contribute to the fitness calculation. Each terminal, that is, *Tile*, provides the

Table 1. GA and GP configuration

Parameter	GA Value	GP Value
Population Size	200	200
Max Generations	1,000	1,000
Crossover Rate	0.70	0.70
Mutation Rate	0.0001	0.25
Elitism Rate	0.05	0.05
Crossover	2-point crossover	standard 1-point crossover
	between integer boundaries	between randomly selected subtrees
Mutation	Randomly selected tile	Randomly generated sub-tree
Max depth	-	7
Min depth	-	2
Terminal	-	Tile
Function	-	TileJoint
Target Size	120 x 100 pixels	120 x 100 pixels
Tile Size	5 x 5 pixels	5 x 5 pixels
Selection	Proportional to fitness	Proportional to fitness
Replacement	Generational replacement	Generational replacement
Termination	Number of generations	Number of generations

index of the selected tile from the tile collection. Unlike GA, GP constructs parse trees of differing height for different individuals. However, for the photomosaic problem, only the first C tiles ($C = 480$ in our experiment), traversed from a parse tree will be used to construct the phenotype which is a photomosaic. If a parse tree has fewer than C terminals, the fitness will still be calculated but the individual will eventually be discarded in the evolutionary process, as the fitness value is worse. As the fitness evaluation is based on equation 1, a value of 1 indicates that the generated photomosaic bears no resemblance to the target image. Therefore a fitness value *closer to 1* is considered as *the worst* which is also been implemented in the GA approach. The GP program was developed using our own RMITGP package. Table 1 presents the parameters used in the GP program. Fig. 2 also shows an example a GP parse tree(chromosome). For clarity, only two tiles are connected to a TileJoint(TJ) function.

5 Results

In our experiment 5 runs were carried out for each of the GA and GP approach. In the context of photomosaic problems the best fitness value remains unknown, since the value of pixels in a photomosaic is not an exact duplicate of the target and a perfect matching between the target image and the generated photomosaic will never occur. Therefore, in this problem, the number of generations is used as the termination condition (as mentioned in Table 1).

Over the course of 1000 generations, the average of 5 runs shows an interesting result. GA exceeds the performance of GP based on the best fitness values

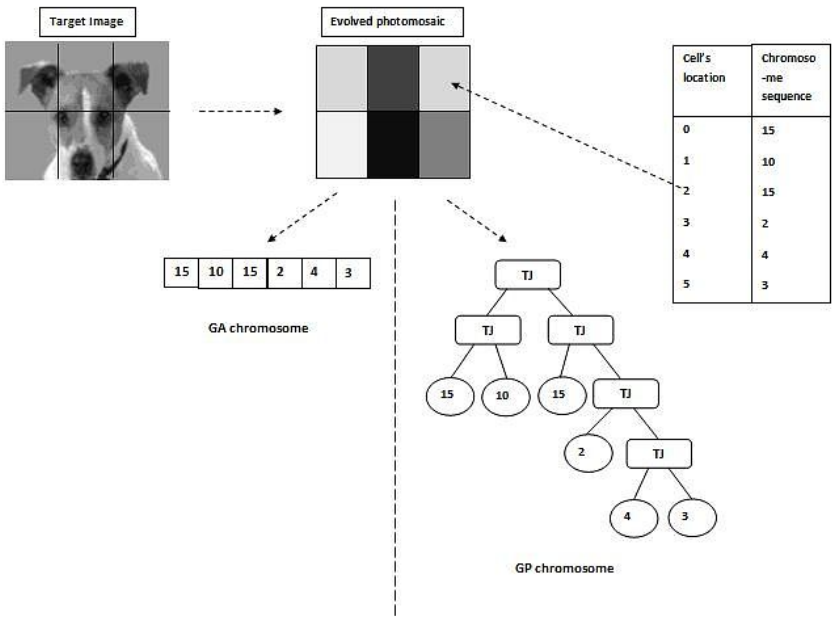


Fig. 2. An example of the GA and GP representations for photomosaic generation

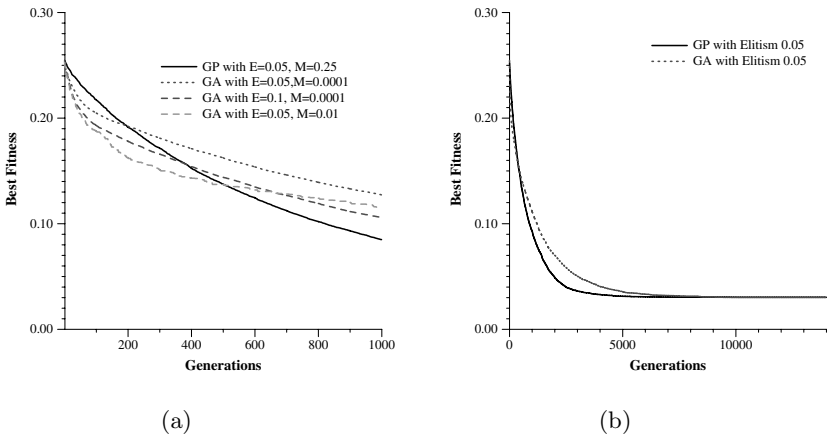


Fig. 3. (a) Results of GP and GA with different elitism rates and mutation rate(E and M represent elitism rate and mutation rate) (b) The performance of GP and GA for 14 000 generations

between generations. However at almost 200 generations, GP starts to converge with GA and thereafter achieves better fitness values than GA.

Based on these preliminary results, we conducted further experiments with the GA program. Using an elitism rate of 0.1 with other parameters remain the same,

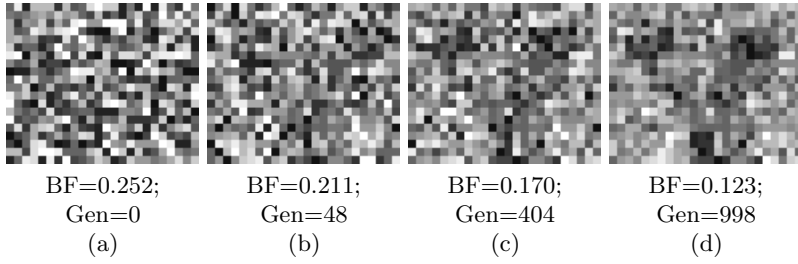


Fig. 4. Sample frames from the GA approach with elitism rate 0.05, BF represents best fitness value while Gen refers to generation

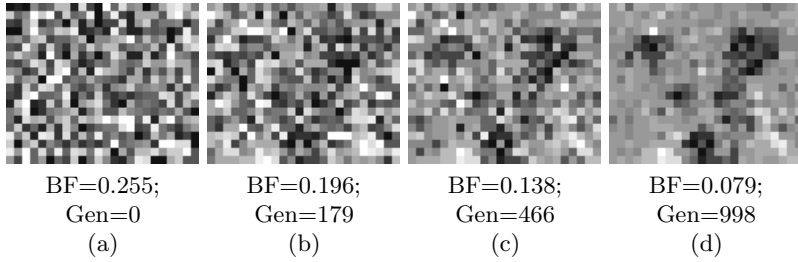


Fig. 5. Sample frames from the GP approach, BF represents best fitness value while Gen refers to generation

we repeated the 5 runs. We found that the new elitism rate further improved GA performance, though again it was eventually outperformed by the GP program. In order to understand more about the GA program, a higher mutation rate (i.e. mutation rate of 0.01) combined with the initial elitism rate of 0.05 have been used in further investigation. Nevertheless, the result did not shows much improvement to the GA performance. Figure 3(a) presents the average best fitness values over 5 runs for the GP program and GA program. The GP program was executed using elitism rate of 0.05 while the GA program uses elitism rates of 0.05, 0.1 and mutation rate of 0.01.

We also looked at the individuals in the GP program and found that in the earlier generations, larger tree size was generated with 1093 nodes consisting of 364 *TileJoint* and 729 *Tile* in the parse tree. However, approaching 1000 generations, we discovered that the size of the parse tree for the best individual was about 730 nodes, comprising 243 *TileJoint* and 487 *Tile* nodes. As the total number of tiles needed is 480, the analysis indicates that the GP program eventually adapts to the desired size of the solution, with a small number of unused tiles for genetic operations.

Finally, for both approaches we investigated the impact of longer runs, specifically to assess convergence. After 14,000 generations, we decided to terminate the runs as the fitness values converged to a stagnant measure, as shown in Figure 3(b). Figure 4 and Figure 5 provide samples of photomosaics generated using GA and GP, respectively.

6 Discussion and Conclusions

Referring to the research questions listed in Section 1, the findings show that, at least for the limited problem size explored, GP is able to produce better photomosaics in fewer generations. Even though the GP and the GA programs eventually converged at almost similar point as shown in Figure 3(b), GP produces better photomosaics than GA. Our initial assumption was that the GA approach would be superior to the GP approach, as the photomosaic generation problem is a fixed-length problem. Our implementation of photomosaic generation adopts a fixed number of cells on a canvas (480 cells in our experiment). Since GA represents a problem in fixed-length chromosomes while in GP the chromosomes are variable-length parse trees, the photomosaic generation problem would appear to be better suited to the GA approach. We expected the GP program to take longer to converge to the result produced by the GA photomosaic program. Our expectation was based on the variation in parse tree height for each individual, as well as the occurrence of an oversupply of tiles in the parse tree, thus lengthening the process of translating genotypes to phenotypes. In reference to Figure 4(d) and Figure 5(d), respectively, the fitness values and the visual qualities of the figures over 1000 generations indicate that the GP produces better photomosaics than the GA program (i.e. closer to target image which can be referred to Fig 2). This might be due to the richer representation brought about by the GP approach. Although only the first 480 terminals are used in a photomosaic construction, the extra terminals in the parse tree are still available for the crossover operation. This could have contributed to the generation of fitter individuals.

On the other hand, for the GA approach, the two-point crossover operator could have been the reason behind its unexpected poor performance. We can only speculate that two-point crossover creates destructive combinations in the evolution process, creating unfit photomosaics. The search space for the GA was about 10^{42} while GP search space was larger with approximately 10^{117} based on the equation from [14]. Despite the search space being massive, GP program was able to narrow down the search to the area of possible best solutions earlier than the GA program.

In terms of development flexibility, to experiment with variations in tile placement strategy in GP (for example, non-grid-based tile placement, in which tiles may be placed anywhere on the canvas) requires an introduction of new terminals, with the extra coding effort localised in the *Draw* function. In contrast, in the GA approach, to implement a new placement strategy requires a major change to the chromosome structure, to include the position information.

As the work presented here is a part of an ongoing study, further experiments and analysis are required to verify the results. Such experiments will involve additional runs and in the context of larger problems of similar nature. However, it is anticipated that future work will be focused on the use of the GP approach for photomosaic generation, given its greater flexibility for a range of tile placement strategies and the representation of the problem.

References

1. Hinterding, R.: Representation, Mutation and Crossover Issues in Evolutionary Computation. In: Proceeding of Congress of 2000 Evolutionary Computation (CEC 2000), vol. 2, pp. 916–923. IEEE Service Center (2000)
2. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Massachusetts (1992)
3. Mitchell, M.: An Introduction to Genetic Algorithms. MIT Press, Massachusetts (1996)
4. Ciesielski, V., Berry, M., Trist, K., D'Souza, D.: Evolution of Animated Photomosaics. In: Giacobini, M., et al. (eds.) EvoWorkshops 2007. LNCS, vol. 4448, pp. 498–507. Springer, Heidelberg (2007)
5. Silvers, R., Hawley, M.: Photomosaic. Henry Holt and Company, Inc., New York (1997)
6. Finkelstein, A., Range, M.: Image Mosaic. In: Hersch, R.D., Andre, J., Brown, H. (eds.) RIDT 1998 and EPub 1998. LNCS, vol. 1375, pp. 11–22. Springer, Heidelberg (1998)
7. Di Blasi, G., Gallo, G., Maria, P.: Smart Ideas for Photomosaic Rendering. In: Proceedings of Eurographics Italian Chapter Conference 2006, Eurographic Association, Catania, Italy (2006)
8. Kim, J., Pellacini, F.: Jigsaw Image Mosaics. ACM Transactions on Graphics (TOG) 21, 657–664 (2006)
9. Park, J.W.: Artistic depiction: Mosaic for Stacktable Objects. In: ACM SIGGRAPH 2004 Sketches SIGGRAPH 2004. ACM, New York (2004)
10. Wijesinghe, G., Mat Sah, S.B., Ciesielski, V.: Grid vs. Arbitrary Placement of Tiles for Generating Animated Photomosaics. In: Proceeding of Congress of 2008 Evolutionary Computation (CEC 2008). IEEE Service Center, Piscataway (2008)
11. Smith, R.E., Goldberg, D.E., Earickson, J.A.: SGA-C: A C-language Implementation of a Simple Genetic Algorithm (1991), <http://citeseer.ist.psu.edu/341381.html>
12. Sinclair, M.C., Shami, S.H.: Evolving simple agents: Comparing genetic algorithm and genetic programming performance. In: IEE Genetic Algorithms in Engineering Systems: Innovations and Applications, pp. 421–426. IEEE Press, New York (1997)
13. Walker, M., Messom, C.H.: A Comparison of Genetic Programming and Genetic Algorithms for Auto-tuning Mobile Robot Motion Control. In: Proceedings of the First IEEE International Workshop on Electronic Design, Test and Applications (DELTA 2002), pp. 507–509. IEEE Press, New York (2002)
14. Ebner, M.: On the search space of genetic programming and its relation to nature's search space. In: Proceedings of the 1999 Congress on Evolutionary Computation, Washington, D.C, July 6–9, vol. 2, pp. 1357–1361. IEEE Press, Los Alamitos (1999)

Parameter Tuning of Real-Valued Crossover Operators for Statistics Preservation

Hiroshi Someya

The Institute of Statistical Mathematics,
4-6-7 Minami-Azabu, Minato-ku, Tokyo, Japan
some@ism.ac.jp
<http://www.ism.ac.jp/~some/>

Abstract. Parameters of real-valued crossover operators have been often tuned under a constraint for preserving statistics of infinite parental population. For applications in actual scenes, in a previous study, an alternative constraint, called *unbiased constraint*, considering finiteness of the population has been derived. To clarify the wide applicability of the unbiased constraint, this paper presents two additional studies: (1) applying it to various crossover operators in higher dimensional search space, and (2) generalization of it for preserving statistics of overall population. Appropriateness of the parameter setting based on the unbiased constraint has been supported in discussion on robust search.

Keywords: Real-coded genetic algorithm, functional specialization hypothesis, statistics preservation, parameter tuning.

1 Introduction

In recent years, efficiency of genetic algorithms based on real-number representation, called real-coded genetic algorithms (RCGAs), has been reported [1]. The representation facilitates statistical analysis of the behavior. Kita et al. have characterized the distribution of a set of individuals by the mean vector and the variance-covariance matrix, and have proposed a guideline, called *statistics preservation*, for designing real-valued crossover operators. This guideline requires that “*distribution of the offsprings generated by crossover operators should preserve the statistics such as the mean vector and the covariance matrix of the distribution of parents well* [2].” If children are distributed narrower than their parents, the crossover operation may lose opportunities to find the optimum without adequate fitness foundation. On the other hand, too wider search would waste computation time in searching hopeless region where selection operation has already abandoned. Therefore sampling new solutions in the region that the parents reside will be an appropriate choice.

For realizing such crossover operations with finite parental population, a constraint, called *unbiased constraint*, has been theoretically derived and has been empirically verified by using a crossover operator UNDX- m [2] with a few parents [3]. To clarify the wide applicability of the unbiased constraint, this paper presents two additional studies: (1) applying it to various crossover operators

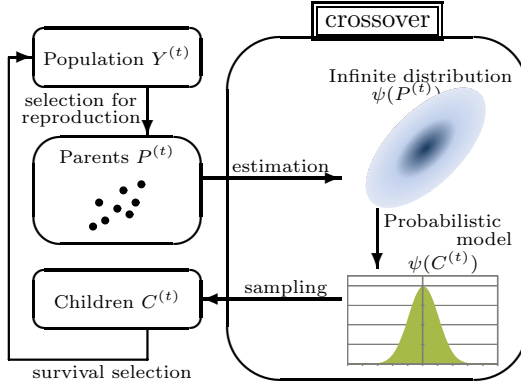


Fig. 1. Roles of crossover operation (from [3] with slight changes)

in higher dimensional search space, and (2) generalization of it for preserving statistics of overall population. Finally, toward realizing robust search, appropriateness of the parameter setting based on the unbiased constraint is discussed.

2 Statistics Preservation of Finite Parental Population

This section briefly reviews the unbiased constraint. To focus on the affect of crossover operation, let consider RCGAs without selection pressure as follows:

- Step 1)* Prepare the initial population $Y^{(0)}$ whose size is ℓ , and set the generation counter $t = 0$.
- Step 2)* Produce r children $C^{(t)}$ using a crossover operator with all individuals in $Y^{(t)}$ as parents $P^{(t)}$, where $\ell \leq r$.
- Step 3)* Choose ℓ individuals from $C^{(t)}$ randomly.
- Step 4)* Replace all individuals in $Y^{(t)}$ with the children survived in *Step 3*.
- Step 5)* Add 1 to t and go back to *Step 2*.

Following the underlying concept of the design guideline, called *functional specialization hypothesis* [4], selection operation should encourage the population in convergence to an optimum on the basis of fitness information and crossover operation should search promising regions provided by selection operation for better individuals by utilizing location information. Since this selection model ignores fitness information, the crossover operation, which behaves like Estimation of Distribution Algorithms [5] using all the individuals in the population as parents, should maintain the statistics of the initial population by playing the following three consecutive roles implicitly (Fig. 1):

- Role 1)* Estimation of the infinite distribution $\psi(P^{(t)})$ of a finite distribution $P^{(t)}$.
- Role 2)* Building a probabilistic model $\psi(C^{(t)})$, which represents desirable infinite distribution to search promising regions around the parents, that preserves the statistics of $\psi(P^{(t)})$ well.
- Role 3)* Producing $C^{(t)}$ by sampling from $\psi(C^{(t)})$.

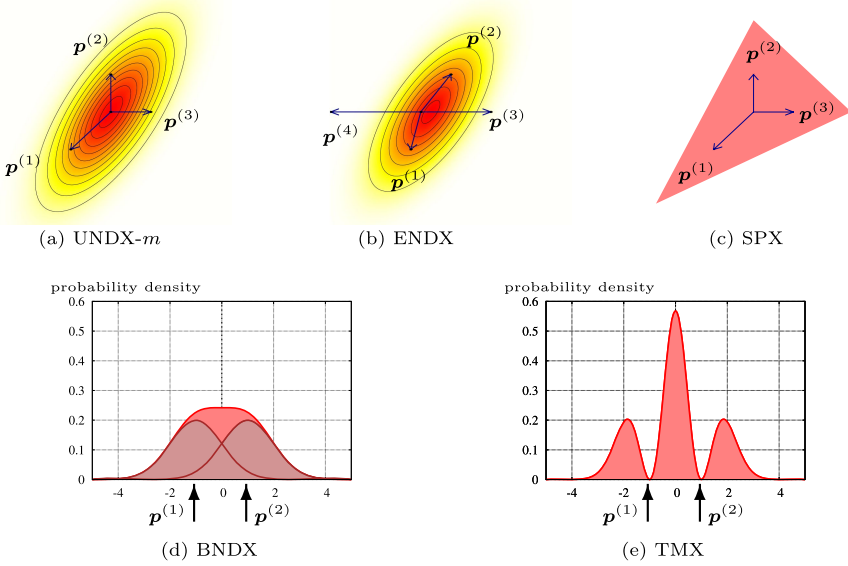


Fig. 2. Examples of children distribution. The three or four parent vectors in (a) $\sim$ (c) are $\mathbf{p}^{(1)} = (-1, -1)$, $\mathbf{p}^{(2)} = (0, 1)$, $\mathbf{p}^{(3)} = (1, 0)$ and $\mathbf{p}^{(4)} = (-3, 0)$. The two parents for (d) and (e) are located at -1 and 1. Each of the two normal distribution in (d) corresponds to the probability density function around $\mathbf{p}^{(1)}$ or $\mathbf{p}^{(2)}$ only. All are tuned under the unbiased constraint.

The symbol $\psi(\cdot)$ indicates the corresponding infinite samples of a finite samples. Let $M[X]$ and $V[X]$ be the mean vector and the variance-covariance matrix of the distribution of a set of individuals X respectively. The first role is achieved by using $M[X] = M[\psi(X)]$ and $V[X] = \frac{k-1}{k}V[\psi(X)]$ where k is the sample size (e.g. [6]). For the rest of the roles, $M[\psi(P^{(t)})] = M[\psi(C^{(t)})]$ and $V[\psi(P^{(t)})] = V[\psi(C^{(t)})]$ are required. Hence the constraint:

$$M[\psi(C^{(t)})] = M[P^{(t)}], \quad (1)$$

$$V[\psi(C^{(t)})] = \frac{\ell}{\ell-1}V[P^{(t)}], \quad (2)$$

called *unbiased constraint*, should be satisfied. This constraint is generally applicable in any-dimensional affine space and is independent of specific crossover operator, as clarified in the sections that follow.

3 Parameter Tuning for Statistics Preservation

This section theoretically determines parameter values of the five crossover operators shown in Fig. 2 for statistics preservation. Let $P = \mathbf{p}^{(1)}, \dots, \mathbf{p}^{(\ell)}$ be a set of constant vectors arbitrarily selected as parents, and $M[P] = \mathbf{g}$ be the center of gravity of them.

3.1 UNDX- m

UNDX- m produces children around the center of gravity of parents as follows:

$$\mathbf{c} = \mathbf{g} + \sum_{j=1}^{\ell-1} \xi^{(j)} \mathbf{d}^{(j)} + \mathbf{q} \quad (3)$$

where $\mathbf{d}^{(j)} = \mathbf{p}^{(j)} - \mathbf{g}$, and $\xi^{(j)}$ are $\ell-1$ instantiations of random variable with the normal distribution $N(0, \sigma_\xi^2)$, where σ_ξ is that tunable crossover parameter that we search for. The $\mathbf{q}$ is determined using one additional parent, where necessary, in the orthogonal complement of the subspace spanned by $\mathbf{d}^{(1)}, \dots, \mathbf{d}^{(\ell-1)}$. For simple discussion, this paper treats $\mathbf{q}$ as zero vector.

UNDX- m satisfies Eq. (1) regardless of the value of σ_ξ as $M[\psi(C)] = \mathbf{g} + \langle \xi \rangle \sum_{j=1}^m \mathbf{d}^{(j)} = \mathbf{g}$ since $\langle \xi \rangle = 0$, where $\langle \cdot \rangle$ denotes the expected scalar value or the expected mean vector. Since $V[\psi(C)] = \langle \xi^2 \rangle \sum_{j=1}^{\ell-1} \mathbf{d}^{(j)} \mathbf{d}^{(j)T}$ where T denotes transposed vector, the relationship between $V[\psi(C)]$ and $V[P]$ when the ℓ -th parent is randomly determined in each producing process is obtained as

$$V[\psi(C)] = \langle \xi^2 \rangle \left\langle \sum_{j=1}^{\ell-1} \mathbf{d}^{(j)} \mathbf{d}^{(j)T} \right\rangle = (\ell-1) \langle \xi^2 \rangle V[P]. \quad (4)$$

Therefore, since $\langle \xi^2 \rangle = \sigma_\xi^2$, children distributed z times wider or narrower than their parents are given by setting the parameter as $\sigma_\xi = \sqrt{\frac{z}{\ell-1}}$.

3.2 Extended Normal Distribution Crossover (ENDX)

ENDX [7] searches along the line segment connecting $\mathbf{p}^{(1)}$ and $\mathbf{p}^{(2)}$ as follows:

$$\begin{aligned} \mathbf{c} &= \mathbf{g}' + \xi \left(\mathbf{p}^{(1)} - \mathbf{g}' \right) + \sum_{j=3}^{\ell} \eta^{(j)} \mathbf{p}'^{(j)}, \\ \mathbf{g}' &= \frac{\mathbf{p}^{(1)} + \mathbf{p}^{(2)}}{2}, \quad \mathbf{p}'^{(j)} = \mathbf{p}^{(j)} - \frac{1}{\ell-2} \sum_{k=3}^{\ell} \mathbf{p}^{(k)}, \quad \eta \sim N(0, \sigma_\eta^2). \end{aligned} \quad (5)$$

Since $\langle \eta \rangle = 0$, the expected mean vector of children is $\mathbf{g}'$. When the order of parents are randomly shuffled in each producing process, Eq. (1) is satisfied as $M[\psi(C)] = \langle \mathbf{g}' \rangle = \mathbf{g}$. As for $V[\psi(C)]$, the first term of the right side of $V[\psi(C)] = \langle \mathbf{c} \mathbf{c}^T \rangle - \mathbf{g} \mathbf{g}^T$ can be written as

$$\langle \mathbf{c} \mathbf{c}^T \rangle = \mathbf{g}' \mathbf{g}'^T + \langle \xi^2 \rangle \left(\mathbf{p}^{(1)} - \mathbf{g}' \right) \left(\mathbf{p}^{(1)} - \mathbf{g}' \right)^T + \langle \eta^2 \rangle \sum_{j=3}^{\ell} \mathbf{p}'^{(j)} \mathbf{p}'^{(j)T}. \quad (6)$$

Substituting the equations:

$$\mathbf{g}'\mathbf{g}'^T = \frac{(\mathbf{p}^{(1)}\mathbf{p}^{(1)} + \mathbf{p}^{(2)}\mathbf{p}^{(2)}) + (\mathbf{p}^{(1)}\mathbf{p}^{(2)} + \mathbf{p}^{(2)}\mathbf{p}^{(1)})}{4}, \quad (7)$$

$$(\mathbf{p}^{(1)} - \mathbf{g}')(\mathbf{p}^{(1)} - \mathbf{g}')^T = \frac{(\mathbf{p}^{(1)}\mathbf{p}^{(1)} + \mathbf{p}^{(2)}\mathbf{p}^{(2)}) - (\mathbf{p}^{(1)}\mathbf{p}^{(2)} + \mathbf{p}^{(2)}\mathbf{p}^{(1)})}{4}, \quad (8)$$

$$\sum_{j=3}^{\ell} \mathbf{p}'^{(j)}\mathbf{p}'^{(j)T} = \sum_{j=3}^{\ell} \mathbf{p}^{(j)}\mathbf{p}^{(j)T} - \frac{\ell^2 \mathbf{g}\mathbf{g}^T - 2\ell(\mathbf{g}'\mathbf{g}'^T + \mathbf{g}\mathbf{g}'^T) + 4\mathbf{g}'\mathbf{g}'^T}{\ell - 2}, \quad (9)$$

into Eq. (6) leads

$$\begin{aligned} \langle \mathbf{c}\mathbf{c}^T \rangle &= \left(\frac{1 + \langle \xi^2 \rangle}{4} - \frac{\langle \eta^2 \rangle}{\ell - 2} \right) \sum_{j=1}^2 \mathbf{p}^{(j)}\mathbf{p}^{(j)T} + \langle \eta^2 \rangle \sum_{j=3}^{\ell} \mathbf{p}^{(j)}\mathbf{p}^{(j)T} \\ &+ \left(\frac{1 - \langle \xi^2 \rangle}{4} \right) \sum_{j=1}^2 \mathbf{p}^{(j)}\mathbf{p}^{(3-j)T} - \frac{\langle \eta^2 \rangle}{\ell - 2} \{ \ell^2 \mathbf{g}\mathbf{g}^T - 2\ell(\mathbf{g}'\mathbf{g}'^T + \mathbf{g}\mathbf{g}'^T) \}. \end{aligned} \quad (10)$$

When the order of parents are randomly shuffled in each producing process, $\langle \mathbf{p}^{(1)}\mathbf{p}^{(2)} \rangle = \langle \mathbf{p}^{(2)}\mathbf{p}^{(1)} \rangle = \frac{1}{\ell-1} (\ell \mathbf{g}\mathbf{g}^T - \langle \mathbf{p}^{(j)}\mathbf{p}^{(j)T} \rangle)$ is formed. Hence

$$\begin{aligned} V[\psi(C)] &= \left\{ \frac{\ell \langle \xi^2 \rangle + (\ell - 2)}{2(\ell - 1)} + \frac{\ell^2 - 4\ell + 2}{\ell - 2} \langle \eta^2 \rangle \right\} \langle \mathbf{p}^{(j)}\mathbf{p}^{(j)T} \rangle \\ &- \left\{ \frac{\ell \langle \xi^2 \rangle + (\ell - 2)}{2(\ell - 1)} + \frac{\ell^2 - 4\ell}{\ell - 2} \langle \eta^2 \rangle \right\} \mathbf{g}\mathbf{g}^T \end{aligned} \quad (11)$$

is obtained¹. In the cases that ℓ is sufficiently large, since

$$\lim_{\ell \rightarrow \infty} \frac{\ell \langle \xi^2 \rangle + (\ell - 2)}{2(\ell - 1)} = \frac{\langle \xi^2 \rangle + 1}{2}, \quad \lim_{\ell \rightarrow \infty} \frac{\ell(\ell - 4) + 2}{\ell - 2} = \lim_{\ell \rightarrow \infty} \frac{\ell(\ell - 4)}{\ell - 2} = \ell - 2, \quad (12)$$

Eq. (11) is simplified as

$$V[\psi(C)] \approx \left\{ \frac{\langle \xi^2 \rangle + 1}{2} + (\ell - 2) \langle \eta^2 \rangle \right\} V[P]. \quad (13)$$

In the other cases, the equation

$$V[\psi(C)] \approx \left\{ \frac{\ell \langle \xi^2 \rangle + (\ell - 2)}{2(\ell - 1)} + (\ell - \delta) \langle \eta^2 \rangle \right\} V[P], \quad 2 \leq \delta \leq 4 \quad (14)$$

¹ This equation is different from the relationship presented in [7]. According to the first author's private advice, he has assumed that parent vectors are independent random variables that satisfy $\langle \mathbf{p}^{(i)}\mathbf{p}^{(j \neq i)T} \rangle = \mathbf{g}\mathbf{g}^T$ unlike in the case of this paper.

would be suitable. Therefore, the σ_ξ and σ_η should satisfy either of the follows:

$$\sigma_\xi = \begin{cases} \sqrt{2\{z - (\ell - 2)\sigma_\eta^2\} - 1} & : \ell \rightarrow \infty \\ \sqrt{\frac{2(\ell-1)\{z - (\ell-\delta)\sigma_\eta^2\} - (\ell-2)}{\ell}} & : \text{otherwise.} \end{cases} \quad (15)$$

3.3 Simplex Crossover (SPX)

SPX [8] produces children within a simplex determined by the tunable parameter ε using uniformly random variables, denoted as $U[\text{lower}, \text{upper}]$, as follows:

$$\mathbf{c} = \mathbf{g} + \varepsilon \left(\mathbf{d}^{(\ell)} + \mathbf{s}^{(\ell)} \right), \quad (16)$$

$$\mathbf{s}^{(j)} = \begin{cases} \mathbf{0} & : j = 1 \\ \nu^{(j-1)} \{ \mathbf{p}^{(j-1)} - \mathbf{p}^{(j)} + \mathbf{s}^{(j-1)} \} & : j = 2 \dots \ell \end{cases} \quad (17)$$

where $\nu^{(k)} \sim U[0, 1]^{\frac{1}{k}}$. Equation (17) is expressed using $\mathbf{r}^{(j)} = \mathbf{p}^{(j-1)} - \mathbf{p}^{(j)}$ as

$$\mathbf{s}^{(j)} = \sum_{h=2}^j \left\{ \left(\prod_{i=h}^j \nu^{(i-1)} \right) \mathbf{r}^{(h)} \right\}, \quad j \geq 2. \quad (18)$$

By using $\langle \nu^{(k)} \rangle = \int_0^1 x^{\frac{1}{k}} dx = \frac{k}{k+1}$ and $\langle \prod_{i=h}^j \nu^{(i-1)} \rangle = \frac{h-1}{j}$,

$$\langle \mathbf{s}^{(\ell)} \rangle = \frac{1}{\ell} \sum_{h=2}^{\ell} (h-1) \mathbf{r}^{(h)} = \frac{1}{\ell} \left\{ \sum_{h=1}^{\ell-1} h \mathbf{p}^{(h)} - \sum_{h=2}^{\ell} (h-1) \mathbf{p}^{(h)} \right\} = -\mathbf{p}^{(\ell)} + \mathbf{g} \quad (19)$$

gives $M[\psi(C)] = \mathbf{g} + \varepsilon (\mathbf{d}^{(\ell)} + \langle \mathbf{s}^{(\ell)} \rangle) = \mathbf{g}$. As for $V[\psi(C)]$, the $\langle \mathbf{s}^{(\ell)} \mathbf{s}^{(\ell)T} \rangle$ in $V[\psi(C)] = \varepsilon^2 (-\mathbf{d}^{(\ell)} \mathbf{d}^{(\ell)T} + \langle \mathbf{s}^{(\ell)} \mathbf{s}^{(\ell)T} \rangle)$ derived using Eq. (19) is expanded as

$$\begin{aligned} \langle \mathbf{s}^{(\ell)} \mathbf{s}^{(\ell)T} \rangle &= \sum_{h=2}^{\ell-1} \sum_{k=h+1}^{\ell} \left\langle \prod_{i=k}^{\ell} \{ \nu^{(i-1)} \}^2 \right\rangle \left\langle \prod_{i=h}^{k-1} \nu^{(i-1)} \right\rangle \mathbf{r}^{(h)} \mathbf{r}^{(k)T} \\ &\quad + \sum_{h=2}^{\ell} \left\langle \prod_{i=h}^{\ell} \{ \nu^{(i-1)} \}^2 \right\rangle \mathbf{r}^{(h)} \mathbf{r}^{(h)T} \\ &\quad + \sum_{h=3}^{\ell} \sum_{k=2}^{h-1} \left\langle \prod_{i=h}^{\ell} \{ \nu^{(i-1)} \}^2 \right\rangle \left\langle \prod_{i=k}^{h-1} \nu^{(i-1)} \right\rangle \mathbf{r}^{(h)} \mathbf{r}^{(k)T}. \end{aligned} \quad (20)$$

Substituting the equation $\langle \prod_{i=a}^{\ell} \{ \nu^{(i-1)} \}^2 \rangle = \frac{a(a-1)}{\ell(\ell+1)}$, derived from $\langle \{ \nu^{(k)} \}^2 \rangle = \int_0^1 x^{\frac{2}{k}} dx = \frac{k}{k+2}$, into the above gives

$$\begin{aligned} \langle \mathbf{s}^{(\ell)} \mathbf{s}^{(\ell)T} \rangle &= \frac{1}{\ell(\ell+1)} \left[\sum_{h=2}^{\ell} \left\{ h \mathbf{r}^{(h)} \right\} \sum_{k=2}^{\ell} k \mathbf{r}^{(k)T} + \sum_{h=2}^{\ell-1} \left\{ -\mathbf{r}^{(h)} \sum_{k=h+1}^{\ell} k \mathbf{r}^{(k)T} \right\} \right. \\ &\quad \left. + \sum_{h=2}^{\ell} \left\{ h \mathbf{r}^{(h)} \sum_{k=2}^h -\mathbf{r}^{(k)T} \right\} \right]. \end{aligned} \quad (21)$$

Since the linear sums of the difference vectors are expressed as $\sum_{k=a \geq 2}^b k \mathbf{r}^{(k)} = a \mathbf{p}^{(a-1)} - b \mathbf{p}^{(b)} + \sum_{k=a}^{b-1} \mathbf{p}^{(k)}$ and $\sum_{k=a \geq 2}^b \mathbf{r}^{(k)} = \mathbf{p}^{(a-1)} - \mathbf{p}^{(b)}$, the relationship

$$V[\psi(C)] = \frac{\varepsilon^2}{\ell + 1} V[P] \quad (22)$$

is obtained². Hence, the parameter value should be tuned as $\varepsilon = \sqrt{z(\ell + 1)}$.

3.4 Bimodal Normal Distribution Crossover (BNDX)

BNDX³ [9] produces two children at one time around each of two parents as

$$\mathbf{c}^{(1)} = \mathbf{p}^{(1)} + \xi \mathbf{d}^{(1)} + \mathbf{q}, \quad \mathbf{c}^{(2)} = \mathbf{p}^{(2)} - \xi \mathbf{d}^{(1)} - \mathbf{q}. \quad (23)$$

When sufficiently many children are produced, the aboves are unified into

$$\mathbf{c} = \mathbf{p}^{(\beta)} + \xi \mathbf{d}^{(1)} + \mathbf{q} \quad (24)$$

where $\mathbf{p}^{(\beta)}$ takes either of the parents in equal probability. The expected mean vector of children obviously equals that of parents. The second-order statistics

$$V[\psi(C)] = (\langle \xi^2 \rangle + 1) V[P] \quad (25)$$

is also easily derived. Hence the parameter value is obtained as $\sigma_\xi = \sqrt{z - 1}$.

3.5 Trimodal Distribution Crossover (TMX)

TMX [10] has been designed with the motivation “do not search already visited regions over and over again.” The probability density function is

$$\begin{aligned} \mathbf{c}^{(1,2)} &= \mathbf{g} \pm \tau \mathbf{d}^{(1)} \pm \mathbf{q}, \\ \tau &\sim T(0, \sigma_\xi^2), \quad T(\mu, \sigma_\xi^2) = N(\mu, \sigma_\xi^2) \cdot \frac{\{(x - \mu)^2 - 1\}^2}{3\sigma_\xi^4 - 2\sigma_\xi^2 + 1}. \end{aligned} \quad (26)$$

When sufficiently many children are produced, Eq. (26) is equivalent to

$$\mathbf{c} = \mathbf{g} + \tau \mathbf{d}^{(1)} + \mathbf{q}. \quad (27)$$

Equation (1) is satisfied since $\langle \tau \rangle = 0$. The relationship on $V[\psi(C)]$ is given as

$$V[\psi(C)] = \left\langle \left(\tau \mathbf{d}^{(1)} \right) \left(\tau \mathbf{d}^{(1)} \right)^T \right\rangle = \langle \tau^2 \rangle V[P]. \quad (28)$$

Thus $15\sigma_\xi^6 - 3(z+2)\sigma_\xi^4 + (2z+1)\sigma_\xi^2 - z = 0$, given from $\langle \tau^2 \rangle = \frac{\sigma_\xi^2(15\sigma_\xi^4 - 6\sigma_\xi^2 + 1)}{3\sigma_\xi^4 - 2\sigma_\xi^2 + 1}$, must be satisfied. By using Cardano’s cubic formula, the followings are obtained;

$$\begin{aligned} \sigma_\xi &= \sqrt{\frac{A + B + z + 2}{15}}, \quad A = (J + K)^{\frac{1}{3}}, \quad B = (J - K)^{\frac{1}{3}}, \\ J &= z^3 - 9z^2 + 87z - 7, \quad K = 5\sqrt{2(3z^4 - 28z^3 + 156z^2 - 24z + 1)}. \end{aligned} \quad (29)$$

² This equation is different from the relationship presented in [8]. In the paper, each parent would be assumed as an independent sample unlike in the case of this paper.

³ BNDX had been called *NDX* until 1997.

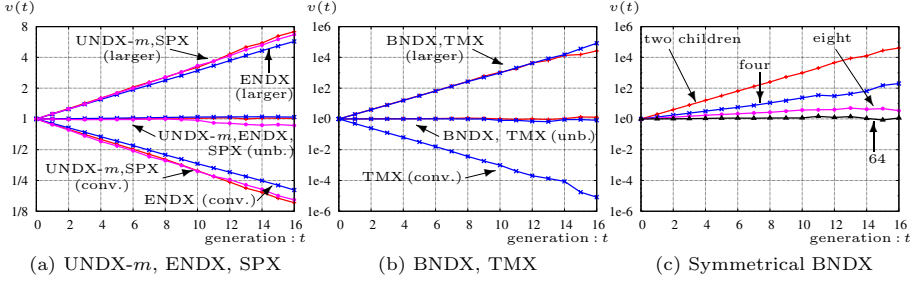


Fig. 3. Evolution of variance, averaged over $1e6$ trials. The *conv.* and the *unb.* mean “conventional” and “unbiased” respectively. The (a) has been obtained by using nine or ten parents for “UNDX- m and SPX” or ENDX, respectively, in eight-dimensional search space. In (b) and (c), two parents have been used in one-dimensional one. For ENDX, the usual value $\sigma_\eta = 0.35/\sqrt{\ell-3}$ [7] and $\delta = 2.5$ have been set.

4 Experimental Studies

4.1 Empirical Verification

The theoretical analyses in the preceding sections are verified by comparing the manners of parameter tuning under the following constraints:

- conventional constraint: $z = 1$, widely used at present, derived in [2, 11];
- unbiased constraint: $z = \frac{\ell}{\ell-1}$, shown in Eq. (2);
- larger constraint: $z = \left(\frac{\ell}{\ell-1}\right)^2$, determined for a geometric progression.

Each single crossover operation, within the selection model presented in Section 2, produces 64 children. The order of the parents is randomly shuffled per each child. For quantitative comparison and simple visualization of the statistics, the amount of change of $n \times n$ matrix $V[Y^{(t)}]$ in which each entry is represented as $\sigma_{Y_{ij}^{(t)}}$ is measured using the scalar $v(t) = \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \sigma_{Y_{ij}^{(t)}} / \sigma_{Y_{ij}^{(0)}}$. Thus, in a logarithmic graph, the dots for $v(t)$ should draw either of the corresponding straight lines: $v(t) = \left(\frac{\ell-1}{\ell}\right)^t$, $v(t) = 1$ and $v(t) = \left(\frac{\ell}{\ell-1}\right)^t$. The experimental results summarized in Fig. 3 (a) and (b) clearly match these theoretical lines, and they attest that the crossover operations under the unbiased constraint preserve the second-order statistics well.

4.2 Effects of Symmetrical Sampling

Whereas the empirical verification, distribution of a small number of children produced by BNDX or TMX following the original definition: Eq. (23) or (26) tends to be too wider under the unbiased constraint. UNDX- m implemented as $\mathbf{c}^{(1,2)} = \mathbf{g} \pm \sum_{j=1}^{\ell-1} \xi^{(j)} \mathbf{d}^{(j)} \pm \mathbf{q}$, as in [2], also behaves in the same way. Figure 3 (c) demonstrates these cases. When such implementations are adopted, considering the identity $M[C] \equiv M[\psi(C)]$ resulting from children symmetrically placed, parameter tuning based on the conventional constraint may be a better choice.

5 Dispersion of Overall Population

In many applications, parents are randomly chosen from larger population in size. This section theoretically analyzes the statistics of the overall population in such case, and discusses appropriate parameter setting for robust search.

5.1 Statistics

Let $\mathbf{G}$ be $M[Y]$, and Z be the overall children produced after several crossover operations in which each satisfies both Eq. (1) and $V[\psi(C)] = zV[P]$. The $M[Z]$ equals $M[Y]$ as $M[Z] = \langle M[C] \rangle = \langle M[P] \rangle = \mathbf{G}$. The second-order statistics of the overall population including L individuals is described as

$$V[\psi(Y)] = \frac{L}{L-1} V[Y] = \frac{L}{L-1} \left(\langle \mathbf{p}^{(j)} \mathbf{p}^{(j)T} \rangle - \mathbf{G} \mathbf{G}^T \right). \quad (30)$$

By using $\langle \mathbf{g} \mathbf{g}^T \rangle = \frac{1}{\ell} \left\{ \frac{L-\ell}{L-1} \langle \mathbf{p}^{(j)} \mathbf{p}^{(j)T} \rangle + \frac{(\ell-1)L}{L-1} \mathbf{G} \mathbf{G}^T \right\}$, that of Z is expressed as

$$\begin{aligned} V[\psi(Z)] &= \langle V[\psi(C)] + (\mathbf{g} - \mathbf{G})(\mathbf{g} - \mathbf{G})^T \rangle \\ &= z \langle \mathbf{p}^{(j)} \mathbf{p}^{(j)T} \rangle + (1-z) \langle \mathbf{g} \mathbf{g}^T \rangle - \mathbf{G} \mathbf{G}^T \\ &= \frac{z(\ell-1)L - \ell + L}{\ell(L-1)} \left(\langle \mathbf{p}^{(j)} \mathbf{p}^{(j)T} \rangle - \mathbf{G} \mathbf{G}^T \right). \end{aligned} \quad (31)$$

In order that $V[\psi(Z)] = V[\psi(Y)]$, the equation:

$$z = \frac{\ell L + \ell - L}{(\ell-1)L} = 1 + \frac{\ell}{(\ell-1)L} \quad (32)$$

must be satisfied. This equation includes all of the two constraints and the *evolution equation*: $V[\psi(Y^{(t+1)})] = \frac{1}{2} \left(1 - \frac{2}{L} + \sigma_\xi^2 \right) V[\psi(Y^{(t)})]$ derived in [12]. By substituting $L \rightarrow \infty$ or $L = \ell$ into Eq. (32), the conventional constraint or the unbiased constraint is derived respectively. In the case of $\ell = 2$, the equation is identified with the evolution equation.

5.2 Parameter Setting for Robust Search

In real-world applications, no one knows before a trial where the optimum exists. Therefore, robustness should be given priority over an outstanding performance in a specific case. However, most crossover operations prefer searching the center of the search space much more than the other [10]. This bias remains even in crossover operations tuned under Eq. (32). When a crossover operation that satisfies Eq. (1), as shown in Section 3, produces children, the distribution of $M[C]$ will follow a normal distribution due to the central limit theorem even if the parents are distributed uniformly. Therefore, the distribution of the overall children will roughly form a roundish normal distribution, as presented in [3, 10]. To flatten the bias curve for robust search, children in each crossover operation should be somewhat widely distributed by setting a larger parameter value than that derived from Eq. (32), such as that determined under the unbiased constraint.

6 Conclusion and Future Work

This paper has presented the two additional studies. In the first one, parameter values of the five crossover operators have been theoretically determined under the unbiased constraint, and they have been empirically verified. In the other one, the generalized version has been derived through the analysis for preserving statistics of overall population, and the appropriateness of the parameter setting based on the unbiased constraint has been supported in the discussion on robust search. They have clarified the wide applicability of the unbiased constraint.

As a future work, comprehensive performance comparison in actual numerical optimization should be performed using crossover operators tuned under various constraints. Since suitable parameter value depends on many factors, such as selection model, landscape of objective function, and purpose of the optimization, both of theoretical analysis and experimental evidence must be important.

Acknowledgments. The author thanks Prof. H. Kita (Kyoto University) and Assoc. Prof. S. Kimura (Tottori University) for their helpful advice on the variance-covariance matrix of UNDX-*m* and that of ENDX respectively. This work was supported in part by the “Function and Induction research project of ROIS” and by the Grant-in-Aid for Science Research (No.17200020, 19700228).

References

1. Herrera, F., Lozano, M.: Special issue on real coded genetic algorithms: Operators, models and foundations. *Soft Computing* 9(4) (2005)
2. Kita, H., et al.: Multi-parental extension of the unimodal normal distribution crossover for real-coded genetic algorithms. In: *Proc. of CEC 1999*, pp. 1581–1587 (1999)
3. Someya, H.: Theoretical parameter value for appropriate population variance of the distribution of children in real-coded GA. In: *Proc. of CEC 2008*, pp. 2722–2729 (2008)
4. Kita, H., Yamamura, M.: A functional specialization hypothesis for designing genetic algorithms. In: *Proc. of SMC 1999*, pp. 579–584 (1999)
5. Lozano, J.A., et al. (eds.): *Towards a New Evolutionary Computation: Advances in the Estimation of Distribution Algorithms*. Springer, Heidelberg (2006)
6. Hoel, P.G.: *Elementary Statistics*, 4th edn. John Wiley & Sons, Inc., Chichester (1976)
7. Kimura, S., et al.: An extension of UNDX based on guidelines for designing crossover operators: Proposition and evaluation of ENDX (in Japanese). *Trans. of the Society of Instrument and Control Engineers* 36(12), 1162–1171 (2000)
8. Higuchi, T., et al.: Theoretical analysis of simplex crossover for real-coded genetic algorithms. In: *Proc. of PPSN VI*, pp. 365–374 (2000)
9. Ono, I., et al.: A genetic algorithm with characteristic preservation for function optimization. In: *Proc. of IIZUKA 1996*, pp. 511–514 (1996)
10. Someya, H., Yamamura, M.: A robust real-coded evolutionary algorithm with toroidal search space conversion. *Soft Computing* 9(4), 254–269 (2005)
11. Kita, H., et al.: Theoretical analysis of the unimodal normal distribution crossover for real-coded genetic algorithms. In: *Proc. of 1998 International Conference on Evolutionary Computation*, pp. 529–534 (1998)
12. Beyer, H.G., Deb, K.: On self-adaptive features in real-parameter evolutionary algorithms. *IEEE Trans. on Evolutionary Computation* 5(3), 250–270 (2001)

Hybrid Particle Swarm Optimization Based on Thermodynamic Mechanism

Yu Wu*, Yuanxiang Li, Xing Xu, and Sheng Peng

State Key Lab. of Software Engineering,
Wuhan University, Wuhan 430072, China
{wy08_whu, yxli62, whuxx84}@yahoo.com.cn
<http://www.skelse.org/>

Abstract. This paper describes a thermodynamic particle swarm optimizer (TDPSO) based on the simple evolutionary equations. Inspired by the minimum free energy principle of the thermodynamic theoretics, a rating-based entropy (RE) and a component thermodynamic replacement (CTR) rule are implemented in the novel algorithm TDPSO. The concept of RE is utilized to systemically measure the fitness dispersal of the swarm with low computational cost. And the fitness range of all particles is divided into several ranks. Furthermore, the rule CTR is applied to control the optimal process with steeply fast convergence speed. It has the potential to maintain population diversity. Compared with the other improved PSO techniques, experimental results on some typical minimization problems show that the proposed technique outperforms other algorithms in terms of convergence speed and stability.

Keywords: particle swarm optimizer, thermodynamic, entropy, swarm diversity, replacement rule.

1 Introduction

Particle swarm optimization (PSO) algorithm is a population-based search method which is originally designed and developed by Eberhart and Kennedy in 1995[1]. The basic idea of PSO is inspired by simulating social behavior and is initialized with a population of random solutions, called particles. By sharing the global information, each particle dynamically adjusts the flight direction according to the discoveries and previous experience of all other companions during the search for promising regions of the landscape.

As a stochastic algorithm, PSO has exhibited successful optimization on well-known numerical test problems [2]. One common problem for PSO on multimodal test problems is the phenomenon of premature convergence to suboptimal solutions. Literature [4] have reported the main reasons on this subject. Fast

* This work was supported in part by the National Natural Science Foundation of China (60773009), the National High Technology Research and Development Program of China (863 Program)(2007AA01Z290) and Natural Science Foundation of Hubei Province of China (2007ABA009).

information flow impels the clustering of particles, where all particles converge to a single point as the speed of the particles is decreased with time. Then there are great difficulties to escape local optima for the PSO algorithm. Consequently, the clustering leads to low diversity with fitness stagnation as an overall result.

In order to overcome the above problems, variant improvements of the PSO algorithm have been proposed during the past decades. In general, they can be categorized into three parts: topological structure [5], parameter tuning or adding [6] [8] [2], and the combination with the other evolutionary computation paradigms [7]. Those improved PSO algorithms mainly concentrate on the third part. In 2003, from the point of view of probability acceptance rule, the Metropolis law of Simulated Annealing (SA) is introduced into PSO algorithm to enhance the local search ability around the optimum [10]. Inspired by SA thoughts, it is feasible that the PSO algorithm can combine with the free energy principle of thermodynamic theoretics.

This paper proposes a new model called thermodynamic particle swarm optimization (TDPSO) algorithm, which simulates the competitive model between energy and entropy in annealing to harmonize the conflicts between selective pressure and population diversity. With the parameter of rating-based entropy (RE), the fitness values are transformed into the interval values, and a new method is defined to measure the fitness dispersal of particles. Instead of the original PSO replacement, a component thermodynamics replacement (CTR) rule is introduced to control optimizing process, in which the original particle with higher free energy component will be replaced by the candidate particle with lower one. Based on the minimal free energy principle of thermodynamics, the free energy component formula utilizes the temperature parameter to regulate the balance between the minimization of the energy (equivalently objective function of minimum problem) and the maintenance of population diversity.

The paper is structured as follows: In section 2 we briefly review the simplification mode on PSO. In section 3 we describe some concepts in TDPSO and the calculation flow of the proposed approach. The simulation settings and results are presented in section 4. Finally, the conclusion is given in section 5.

2 The Simplified Model of Particle Swarm Optimization

Making reference to the basic PSO model [8], each individual is treated as a volume-less particle in the D-dimensional search space. The position and velocity vectors of the i th particle in the D-dimensional search space are denoted as $\vec{X}_i(t) = (x_{i1}(t), \dots, x_{iD}(t))$ and $\vec{V}_i(t) = (v_{i1}(t), \dots, v_{iD}(t))$. The best previous position of the i th particle up to time t is recorded as $\vec{P}_i(t)$, which is also called pbest. And the global best position in the population is represented as $\vec{G}(t)$ called gbest. The velocity and position update equations are as follows:

$$v_{ij}(t+1) = \omega v_{ij}(t) + c_1 \gamma_1 (p_{ij}(t) - x_{ij}(t)) + c_2 \gamma_2 (g_j(t) - x_{ij}(t)) \quad (1)$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \quad (2)$$

Where the inertia weight ω is a scaling factor controlling the influence of the old velocity; c_1 and c_2 are positive constants known as "cognitive" and "social" coefficients respectively; γ_1 and γ_2 are two different random numbers sampled in the range $[0,1]$ separated. The velocity components of the particle v_{ij} are limited to a maximum allowable modulus $V_{\max}$.

Through carefully analyzing the PSO biological model, and evolution iterative equation (1) (2), It is easily found that particle velocity concept is not a prerequisite for the evolution process. The literature [11] has proved this foundation and concluded the simplified optimal equation without the velocity as follows

$$x_{ij}(t+1) = \omega x_{ij}(t) + c_1 \gamma_1 (p_{ij}(t) - x_{ij}(t)) + c_2 \gamma_2 (g_j(t) - x_{ij}(t)) \quad (3)$$

3 The Model of Thermodynamic Particle Swarm Optimization

While the tournament mechanism is introduced into TDPSO algorithm, two crucial issues are needed to resolve. The one is how to define entropy which quantificationally measures population energy diversity at low computational cost; the other is how to design the rule CTR which is followed the minimal free energy principle with the approximate steepest descent ability.

3.1 The Principle of Minimum Free Energy

In thermodynamics and statistical mechanics, the principle of minimum free energy is followed through all transition of the system at each temperature [9]. A closed system, initially at high temperature and disordered, adapts itself to approximately reach thermodynamic equilibrium at any temperature. And the system achieves equilibrium when its free energy seeks the minimum. In thermodynamics, the free energy formula is defined as:

$$F = E - HT \quad (4)$$

Where E is the energy of the system, H is its entropy, and T is the temperature which determines the relative weights between energy and entropy in the competition. The concept about entropy is proposed to quantificationally measure the energy dispersal of particles in the system.

3.2 Measurement of Swarm Diversity

The concept of swarm diversity has many different viewpoints of paraphrasing. All of them can be concluded to two aspects: diversity in code space or in object space. At present, most literatures related to swarm diversity are based on the first one. According to the analysis for related test data [13,14], it can be seen that high swarm diversity in object space has improved global optima probability with low computational cost. Therefore, the entropy is deliberately designed

to measure swarm energy dispersal in the solution space. In other words, the measurement of swarm energy diversity is equivalent to analyze the swarm fitness dispersal in the solution space. TDPSO algorithm dynamically conducts a statistical survey on fitness value range and defines a parameter called rating-based entropy (RE). The statistical data come from all the particles generated so far, and the range is divided into several ranks. The calculation of RE can systematically measure the fitness dispersal of population in different ranks with low computational cost.

Suppose that S represents the search space and $f(x)$ an objective function, we take N particles to compose the swarm population which remarks as $POP(t) = \{X_1(t), X_2(t), \dots, X_N(t)\}$ at iterative step t .

Definition 1 (Primary Energy of particle). For each particle, the primary energy of individual is defined as

$$PE(X_r) = \begin{cases} f(X_r); & \text{for minimum problem} \\ -f(X_r); & \text{for maximum problem} \end{cases}.$$

Definition 2 (Fitness Rank). At the iteration t , the fitness range is delimited by a lower bound l_i and an upper bound u_i in the search process: $RG(t) = [l_t, u_t]$. Note that the fitness range is a primary energy range and conducts the statistical survey on the basis of all the particles generated so far.

The range is divided into K ranks marked as $\mathfrak{R}_0, \mathfrak{R}_1, \dots, \mathfrak{R}_{K-1}$, which satisfy, $\bigcup_{i=0}^{K-1} \mathfrak{R}_i = RG(t)$, $\mathfrak{R}_i \cap \mathfrak{R}_j = \emptyset, i \neq j$ and $0 \leq i, j \leq K-1$. And the subset range of $\mathfrak{R}_i$ is defined as $\mathfrak{R}_i = (\frac{\alpha^{i-1}-1}{\alpha^{K-1}-1} * (u_t - l_t), \frac{\alpha^i-1}{\alpha^{K-1}-1} * (u_t - l_t) + l_t] \cap [l_t, u_t]$. In the above definitions, α denotes proportional factorial constant.

For every particle satisfies $PE(X_r) \in \mathfrak{R}_i$, then we shall say that the particle X_r belongs to the rank i th at the iterative step t .

Definition 3 (Rating-based Entropy of Swarm). Assume the amount of the particles at rank i th of the range $RG(t)$ is expressed as n_i . Therefore, the rating-based entropy of the swarm is defined as $H_{POP} = - \sum_{i=0}^{K-1} \frac{n_i}{N} \log_k \frac{n_i}{N}$.

The rating-based entropy is introduced to measure the swarm fitness dispersal at low computational cost. We can prove the entropy is calculated as 0 when all particles of the swarm locates at the same level, or as the big number 1 when the particle number of each rank is same ($n_0 = n_1 = \dots = n_{K-1}$). Therefore, the entropy with small value expresses energy gather of particles in the solution space, and with large value means high energy diversity of the swarm.

3.3 Thermodynamics Replacement Rule

PSO is the only evolutionary algorithm that does not implement survival of the fittest [12]. Different from genetic algorithms, all particles newly generated in PSO are kept as members of the next new population through the evolutionary course. A lot of research on replacement rule has been done in some literatures.

The simulated annealing-particle swarm optimization (SA-PSO) algorithm [10] utilizes the acceptance of Metropolis rule by probability to enhance the local search ability around the optimum. In the thermodynamical genetic algorithm (TDGA) [13], the greedy thermodynamical replacement (GTR) rule is adopted to minimize the free energy of the population. In order to reduce the complexity of the replacement, Ying [14] introduces a component thermodynamical replacement (CTR) rule. In view of the success that the rule CTR keeps stability and improves the computational efficiency of TDGA, this paper does some study to merges PSO with the CTR.

Instead of the original PSO replacement, the rule CTR assigns the free energy of the whole swarm to each particle as the free energy component. Then the rating-based entropy of the swarm is also split into each particle's one calculated as $H = -\log_k(n_i/N)$. The primary energy gives the description to the virtues or defect degree of each particle. With the application of normalized primary energy in the fitness range, it is proposed as the concept of relative energy.

Definition 4 (Relative Energy of Particle). For each particle $PE(X_r) \in \mathfrak{R}_i$, the relative energy of the particle is defined as $RE(X_r) = \frac{i}{K-1}$. Note that the particles at the same rank should have the same relative energy.

Definition 5 (Free Energy Component). Assume the particle belongs to the i th rank at the iteration t , then the free energy component at temperature T is defined as: $F_c(X_r) = RE(X_r) - T * H$.

It is proved by Ying [14] that the swarm free energy is equivalent to the mean value of all particles' free energy component in the population. There are some differences of the rule CTR between this paper and Ying's, for example: the number of new particles generated, the formula definitions of $\mathfrak{R}_i$ and H , especially, the various comparative objects and the producing way of next new population. The replacement rule proposed in TDPSO is described as follows.

The Process of CTR Rule

program Inflation

Input: the fitness range at t iteration---RG(t),
the current temperature T ,
the population at t iteration---POP(t);

Output: the population at $t+1$ iteration---POP($t + 1$)

begin

- 1) Produce N new particles to form the subpopulation SubPOP(t), in which each new particle updates according to the simplification equation (3). And calculate the primary energy of these new particles;
- 2) Expand RG($t + 1$) based on the known RG(t) and the primary energy of all new particles;
- 3) Calculate the free energy component of each particle including POP(t) and SubPOP(t);
- 4) Compare the free energy component between the new particle and its previous one. If the former is less, then the

previous particle is replaced by the new one; otherwise
the previous one is as the member of $POP(t + 1)$.
end.

3.4 Outline of TDPSO

We give the flow chart of the TDPSO algorithm showed in Fig.1. Including the temperature parameter is decremented at each *MarL* iterations. The declining schedule is determined by $T_\mu = T_0/(1 + \mu)$, where μ is the declining- time.

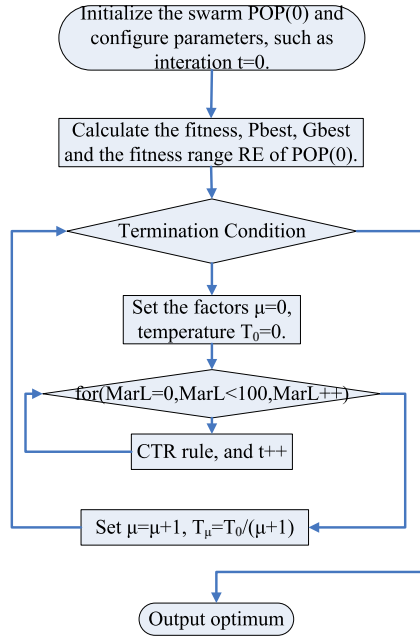


Fig. 1. The Flow of TDPSO algorithm

4 Experimental Setting and Results

In this section, we apply Basic-PSO, PSO-LDW [2], SAPSO, and TDPSO to solve some typical minimization problems in Tab. 1. The initialization range of particles is equivalent to the search range of each function. The first two functions are unimodal whereas the next functions are multimodal.

In the following experiments, all algorithms use the uniform termination condition, common parameters are as follows: the population size $N = 80$, $c_1 = c_2 = 2$. The inertia weight ω of Basic-PSO is generated randomly at each iteration in $[0, 1]$, while in the other compared algorithm the parameter is linearly decreased from 0.9 to 0.4. Except for TDPSO, $V_{\max} = 50\%$ of the search

Table 1. The Benchmark Functions

Fun.	Mathematical Representation	Dim(d)	Range	Optimal
Sphere f_1	$\sum_{i=1}^d x_i^2$	30/50	$[-100, 100]$	0
Rosenbrock f_2	$\sum_{i=1}^d (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$	30/50	$[-100, 100]$	0
Rastrigin f_3	$\sum_{i=1}^d (x_i^2 - 10 \cos(2\pi x_i) + 10)$	30/50	$[-10, 10]$	0
Griewank f_4	$\frac{1}{4000} \sum_{i=1}^d (x_i)^2 - \prod_{i=1}^d \cos(\frac{x_i}{\sqrt{i}}) + 1$	30/50	$[-600, 600]$	0
Schaffer f_5	$0.5 + \frac{(\sin \sqrt{x^2 + y^2})^2 - 0.5}{(1.0 + 0.001(x^2 + y^2))^2}$	2	$[-100, 100]$	0
Ackely f_6	$e + 20 - 20 \exp(-0.2 \sqrt{\frac{1}{d} \sum_{i=1}^d x_i^2}) - \exp(\frac{\sum_{i=1}^d \cos(2\pi x_i)}{d})$	30/50	$[-32, 32]$	0

range in the other optimization. In SAPSO, the initial annealing temperature $T_0 = 100$, the annealing operation is set to $T_{t+1} = \lambda T_t$, where $\lambda = 0.96$. In TDPSO, $T_0 = 100$, $K = 10$ and $\alpha = 2$. All results presented in this paper are averages of 50 repeated runs. The experiments are performed on a computer with Intel Pentium-4 CPU 3.0GHz and 1GB of RAM, by using the visual C++ compiler.

To check whether the algorithms converge to global optimum or not, they are run until either the average optimum value is less than 10^{-8} for function f_1, f_3, f_4, f_5 and 10^{-5} for f_2, f_6 or until a maximum of 3000 Iteration. Tab. 2 and 3 summarizes the results from the conducted experiments. The hyphen(-) mark indicates that none of the runs reached the optimal value. Except for the Rosenbrock function which contains a huge number of local minima, TDPSO converges for all the 50 runs in Tab. 3. Furthermore, the mean number of function evaluations required for convergence in TDPSO tends to be less in the comparison.

Fig.2 show the convergance curves of different optimizations on function f_2, f_3, f_5, f_6 at early stage. Except for the Schaffer function,TDPSO with prominent advantage outperforms the other three optimiazation for all the benchmark functions. By studying the curves in all the figures, it is easy to see that the TDPSO converges quickly under all cases but will slow its convergence speed down when reaching the optima.

For the Sphere, Schaffer and Ackley function $[f_1, f_5, f_6]$, all the compared optimization can be converged to the optima in 30 dimension. Since function f_1 is the easiest problem, PSO-LDW and SAPSO have better ability of improving the solution quality at the end run than Basic-PSO algorithm, and yet TDPSO searches a most precise solution. The function f_5 is designed to trap algorithms searching for local minima. We can observe that PSO-LDW and SAPSO have stronger globe search ability at the beginning than TDPSO in Fig.2(c). However,

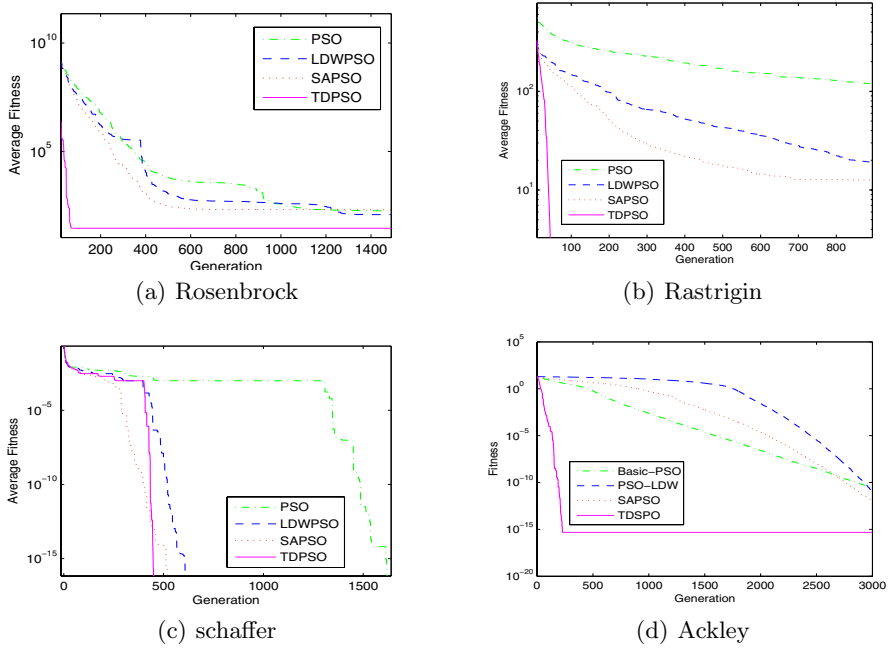


Fig. 2. Experimental results on functions in 30 dimension

Table 2. Average optimum value and standard deviation at the end of 3000 iterations

30 Dimension					
Opt.	f_1	f_2	f_3	f_4	f_6
Basic-PSO	$2.13e-15$	$1.87e+02$	$4.56e+01$	$1.72e-01$	$2.74e-11$
	$9.91e-16$	$2.29e+02$	$1.19e+01$	$6.50e-02$	$1.40e-10$
PSO-LDW	$8.58e-39$	$1.19e+02$	$7.95e+00$	$1.94e-02$	$1.22e-11$
	$2.16e-38$	$1.01e+02$	$2.77e+00$	$1.85e-02$	$8.60e-12$
SAPSO	$1.87e-48$	$1.86e+02$	$8.95e+00$	$9.60e-03$	$8.63e-12$
	$5.15e-48$	$5.60e+01$	$1.40e+00$	$9.48e-03$	$1.03e-11$
TDPSO	$0.00e+00$	$2.89e+01$	$0.00e+00$	$0.00e+00$	$4.77e-16$
	$0.00e+00$	$1.09e-02$	$0.00e+00$	$0.00e+00$	$1.62e-15$
50 Dimension					
Opt.	f_1	f_2	f_3	f_4	f_6
Basic-PSO	$5.26e-04$	$9.35e+03$	$2.70e+02$	$3.10e+00$	$1.33e+00$
	$5.06e+01$	$7.04e+03$	$7.61e+01$	$9.82e-01$	$5.75e+00$
PSO-LDW	$1.94e-22$	$4.37e+02$	$1.67e+02$	$1.99e-02$	$2.10e-04$
	$2.71e-22$	$5.37e+02$	$5.56e+01$	$1.47e-02$	$4.20e-04$
SAPSO	$1.04e-23$	$2.26e+02$	$1.28e+02$	$1.06e-02$	$2.50e-11$
	$5.22e-24$	$1.77e+02$	$4.12e+01$	$7.60e-03$	$1.73e-12$
TDPSO	$0.00e+00$	$4.89e+01$	$0.00e+00$	$0.00e+00$	$6.65e-16$
	$0.00e+00$	$4.48e-02$	$0.00e+00$	$0.00e+00$	$1.98e-15$

Table 3. Number of runs(out of 50) to optimality and corresponding mean number of function evaluation on benchmark functions

30 Dimension						50 Dimension					
Opt.	f_1	f_2	f_3	f_4	f_6	Opt.	f_1	f_2	f_3	f_4	f_6
Basic-PSO	50 142644	0 —	0 —	0 —	50 15230	Basic-PSO	0 —	0 —	0 —	0 —	0 —
PSO-LDW	50 144339	0 —	0 —	11 33932	50 18945	PSO-LDW	50 160050	0 —	0 —	13 62521	5 19164
SAPSO	50 218933	0 —	0 —	16 42433	50 10459	SAPSO	50 216932	0 —	0 —	14 76474	50 12749
TDPSO	50 9467	0 —	50 9841	50 11220	50 9779	TDPSO	50 9158	0 —	50 8485	50 13249	50 9655
Fun.	Dim.	Basic-PSO		PSO-LDW		SAPSO		TDPSO			
f_5	2	50 33329		50 34326		50 25709		50 18358			

for this function, the function evaluation times are shown a minimum by TDPSO in Tab. 3. In the function f_6 , when the dimension is set to 30, most algorithms seek the optimum of the similar precision. The magnitudes of average best fitness and standard deviation are basically the same. Nevertheless, TDPSO presents same performance whatever the dimension is.

On the Rosenbrock, Rastrigin and Griewank benchmark functions [f_2, f_3, f_4], the TDPSO is much stronger optimizer than the other relevant PSO. Although all algorithms fail to find the optimum for the function f_2 , TDPSO finds the best solutions. In function f_3 , the TDPSO algorithm simply finds the global optimum more often whatever the dimension is, and its best fitness value obtained is several magnitudes better than the other three optimizers. For function f_4 , although the Basic-PSO, PSO-LDW and SAPSO obtain the optimum at several times, TDPSO reflects the great stability without standard deviation at each run.

In conclusion, the figures of convergence curves represent that TDPSO has great ability of improving the solution quality with steeply fast convergence speed. And the precision of solution, the stability of this algorithm is respectively showed in Tab. 2 and 3 whatever the dimension is.

5 Conclusion

This paper has presented a novel particle swarm optimization TDPSO algorithm, which is based on the simplified updating formula and the thermodynamical strategy. The performance of this algorithm has been extensively investigated by experimental studies of six non-linear functions.

Inspired by the free energy principle of the thermodynamic theoretics, the definition of RE is utilized to systemically measure the fitness dispersal among all particles in different ranks with low computational cost. Different from the original PSO replacement, the rule CTR simulates the competitive mechanism between

energy and entropy in the principle of free energy. The whole swarm free energy is assigned to each particle as the free energy component in this rule. It is propitious to keep stability and to improve the computational efficiency of TDPSO.

The experimental results illustrate that the TDPSO outperforms Basic-PSO, PSO-LDW, and SAPSO for the complex optimization problems, especially for the multimodal functions in a high dimension space. It has much smaller number of function evaluations and faster convergence rates than some other PSO algorithms.

References

1. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: IEEE International Conference on Neural Network, pp. 1942–1948. IEEE Press, Perth (1995)
2. Shi, Y.H., Eberhart, R.: A modified particle swarm optimizer. In: IEEE International Conference on Evolutionary Computation, pp. 69–73. IEEE Press, Anchorage (1998)
3. Clerc, M., Kennedy, J.: A The particle swarm explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation* 6, 58–73 (2002)
4. J. Riget and J. S. Vesterstrøm: A diversity-guided particle swarm optimizer Cthe arPSO. Dep. of Computer Science, University of Aarhus (2002)
5. Kennedy, J.: SmallWorlds and Mega-Minds: Effects of Neighborhood Topology on Particle Swarm Performance. In: The Congress of Evolutionary Computation, vol. 3, pp. 1931–1938. IEEE Press, Los Alamitos (1999)
6. Eberhart, R., Shi, Y.H.: Comparing Inertia Weights and Constriction Factors in Particle Swarm Optimization. In: IEEE International Conference on Evolutionary Computation, pp. 84–88. IEEE Press, Piscataway (2000)
7. Løvbjerg, M., Rasmussen, T.K., Krink, T.: Hybrid Particle Swarm Optimiser with Breeding and Subpopulations. In: 3rd Genetic and Evolutionary Computation Conference. ACM Press, New York (2001)
8. Shi, Y.H., Eberhart, R.: Empirical study of particle swarm optimization. In: IEEE International Conference on Evolutionary Computation, pp. 101–106. IEEE Press, Los Alamitos (1999)
9. Kennedy, J.: The particle swarm: social adaptation of knowledge. In: IEEE International Conference on Evolutionary Computation, pp. 303–308. IEEE Press, Piscataway (1997)
10. Shang, G., Jingyu, Y.: Particle Swarm Optimization based on the Ideal of Simulated Annealing Algorithm. *Computer Applications and Software* 1, 104–110 (2005)
11. Wang, H., Zhishu, L.: A Simpler and More Effective Particle Swarm Optimization Algorithm. *Journal of Software* 18(4), 861–868 (2007)
12. Eberhart, R., Shi, Y.H.: Comparison between genetic algorithms and particle swarm optimization. In: Annual Conference on Evolutionary Programming. IEEE Press, San Diego (1998)
13. Mori, N., Yoshida, J., Tamaki, H.: A thermodynamical selection rule for the genetic algorithm. In: IEEE International Conference on Evolutionary Computation, pp. 188–192. IEEE Press, Piscataway (1995)
14. Weiqin, Y., Yuanxiang, L.: A thermodynamical selection rule for the genetic algorithm. In: Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2007. LNCS, vol. 4490, pp. 997–1004. Springer, Heidelberg (2007)

Multiagent Evolutionary Algorithm for T-coloring Problem^{*}

Jing Liu, Weicai Zhong, and Jinshu Li

Institute of Intelligent Information Processing, Xidian University, Xi'an, China
neouma@163.com

Abstract. With the properties of T-coloring problems in mind, multiagent systems and evolutionary algorithms are integrated to form a new algorithm, Multiagent Evolutionary Algorithm for T-coloring (MAEA-T-coloring). We studied the generalization of classical graph coloring model, and focused our interest in the restricted T-coloring. An agent in MAEA-T-coloring represents a candidate solution to T-colorings. All agents live in a latticelike environment, with each agent fixed on a lattice-point. In order to increase energies, they compete or cooperate with their neighbors using their knowledge. Experiments on large random instances of T-colorings show encouraging results about MAEA-T-coloring.

1 Introduction

Graph coloring Problem (GCP) is an outstanding concept in graph theory. It is one of the most studied *NP-hard* problems and can be defined informally as follows ^[1]. Given an undirected graph, one wishes to color the vertexes of the graph with a minimal number of colors in such a way that two colors assigned to two adjacent vertexes must be different, i.e., they must have a minimal distance greater than zero ^[2]. Several extensions of GCP exist allowing more applications to be embraced. Such as frequency assignment, traffic phasing, job assignment and so on. T-coloring problem is an important extensions allowing one to model the frequency assignment problem ^[3] ^[4], and the general GCP is just a special case of T-colorings.

Unfortunately, due to the difficulty and complexity of this problem, there has rather little research on T-colorings in recent years. Some classical methods were proposed several years ago. Such as tabu search, *Dsatur* ^[5] and simulated annealing for T-colorings. Until now, there is no benchmark available for evaluating and comparing different algorithms for T-colorings.

Agent-based computation has been studied for several years in the field of distributed artificial intelligence, and has been widely used in many other branches of computer science. There has been a considerable research effort about this, such as

^{*} This work was supported by the National Natural Science Foundations of China under Grant 60502043, 60872135, and 60602064, the Program for New Century Excellent Talents in University of China under Grant NCET-06-0857, the National High Technology Research and Development Program ("863" program) of China under Grant 2006AA01Z107, and the Natural Science Research Project of Shaanxi, China.

numerical optimization, combinatorial optimization, constraint satisfaction problems, constrained layout optimization, and multicast routing problems^[6].

In this paper, we combine the multiagent system and evolutionary algorithm (EAs) to form a new algorithm for T-coloring Problem (MAEA-T-coloring). In the next section, T-coloring and the mathematic model are introduced, and then we described MAEA-T-coloring and how we have applied it to the random instances. In Section 3, the experimental results compared with the classical algorithm *Dsatur* is presented. Finally, Section 4 makes a short conclusion for this article.

2 Multiagent Evolutionary Algorithm for T-coloring Problem

2.1 The T-coloring Problem

In T-colorings, the forbidden separation distance for colors assigned to adjacent vertexes is no longer limited to the singleton $\{0\}$, but may be any set of positive integers. Let us consider the graph $G=(V, E)$, where $V=\{v_1, v_2, \dots, v_n\}$ represents the set of vertexes and $E=\{e_{ij} \mid \text{an edge between } v_i \text{ and } v_j\}$ denotes the set of edges. A collection of sets $T=\{T_{ij} \subset \mathbf{IN} \mid \text{for each } e_{ij} \in E\}$ is now defined to determine for each edge e_{ij} the color separations that are not allowed between vertex v_i and v_j . Where $\mathbf{IN}$ is the set of integers. That is to say, each T_{ij} is a set of unsigned integers such as $\{0, 1, 5, 8\}$, and the color constraint is in (1), where the color of each vertex is coded as an integer, and $c(v_i)$ is the color assigned to vertex v_i .

$$\forall e_{ij} \in E, \quad |c(v_i) - c(v_j)| \notin T_{ij} \quad (1)$$

The separation of colors assigned to two adjacent vertexes v_i and v_j must be different from those of T_{ij} . A T-coloring of a graph is a partition of V in different color classes $C_1, C_2, \dots, C_k$, such that the property (1) is verified for each edge of G . The *chromatic number* $\chi_T(G)$ corresponds to the minimum number of different color values used to color G . The *span* of a T-coloring is the difference between the smallest and the highest color values needed to obtain the T-coloring of G . The T-colorings problem is to determine the *minimum span* $sp_T(G)$ for all the possible colorings of G ^[7].

If each T_{ij} is a set of consecutive integers of the form $T_{ij}=\{0, 1, 2, \dots, t_{ij}-1\}$, the *restricted T-coloring problem* can be defined where constraint (1) becomes:

$$\forall e_{ij} \in E, \quad |c(v_i) - c(v_j)| \geq t_{ij} \quad (2)$$

where t_{ij} is a fixed positive number associated to each edge e_{ij} .

In the rest of this paper, we limit ourselves to the *restricted T-coloring problem*. It is easy to see that the graph coloring problem is a special case of the T-colorings problem where all $T_{ij}=\{0\}$.

2.2 The Agent Used in T-coloring Problem

In general, an agent is a physical or virtual entity essentially having the following properties^[8]: (a) it is able to live and act in the environment; (b) it is able to sense the

local environment; (c) it is driven by certain purposes and (d) it has some reacting behaviors. Multiagent systems are computational systems in which several agents interact or work together in order to achieve purposes. As can be seen, the meaning of agent is very comprehensive, and what an agent represents is different for different problems. In general, four elements should be defined when multiagent systems are used to solve problems. The first is the meaning and the purpose of each agent. The second is the environment in which all agents live. Since each agent has only local perceptivity, so the third is local environment. The last is the behaviors that each agent can take to achieve the purpose^[9]. Three definitions are given as follows:

Definition 1. An agent for T-coloring, *agent*, represents an element in the search space, *S*, and each agent can be expressed as an integer permutation as follows:

$$\text{agent} = (n_1, n_2, \dots, n_N), \quad n_i \in \mathbb{IN} \quad (3)$$

Where *N* is the number of vertexes of *G*.

The domain for each variable is finite and discrete, thus the elements can be numbered by natural numbers. When all domains are transformed into the sets of natural numbers, the solutions of the problem shall present specific characteristics. So we defined the **permutation agent** and the **solution agent**.

All permutation agents are represented by a permutation of 1, 2, ..., *N*, which means the order we color the vertices of *G*. All solution agents are represented by sets of natural numbers, which means the solution we colored the vertices of *G*. Each solution agent is the color result according to one permutation agent.

Definition 2. As what we are concerned about is the *minimum span* of *G* in T-colorings, we defined the energy of an agent as the negative value of the *span* of *G*:

$$\forall \text{agent} \in \mathbb{S}, \text{Energy}(\text{agent}) = -\text{span}_T(G) \quad (4)$$

The purpose of an agent is to maximize the energy by the behaviors it takes. As can be seen, the higher the energy is, the smaller the *span* is, which means the closer to the optima the solution is.

Since each agent must record some information, it can be represented by the following structure:

Agent = Record

P: *P* ∈ *S*;

E: The energy of the agent, *E*=*Energy(P)*;

SL: The flag for the self-learning behavior, which will be defined later. If *SL* is *True*, the self-learning behavior can be performed on the agent, otherwise, cannot;

End.

Definition 3. All of the agents live in a latticelike environment, L , which is called an agent lattice. The size of L is $L_{size} \times L_{size}$, where L_{size} is an integer. Each agent is fixed on a lattice-point and can only interact with the neighbors. Suppose that the agent located at (i, j) is represented as $L_{i,j}$, $i, j=1, 2, \dots, L_{size}$, then the neighbors of $L_{i,j}$, $Neighbors_{i,j}$, are defined as follows:

$$Neighbors_{i,j} = \{L_{i',j}, L_{i,j'}, L_{i'',j}, L_{i,j''}\} \quad (5)$$

Where $i' = \begin{cases} i-1 & i \neq 1 \\ L_{size} & i = 1 \end{cases}$, $j' = \begin{cases} j-1 & j \neq 1 \\ L_{size} & j = 1 \end{cases}$, $i'' = \begin{cases} i+1 & i \neq L_{size} \\ 1 & i = L_{size} \end{cases}$, $j'' = \begin{cases} j+1 & j \neq L_{size} \\ 1 & j = L_{size} \end{cases}$.

The agent lattice can be represented as the one in Fig.1. Each circle represents an agent, the data represent the position in the lattice, and two agents can interact with each other if and only if there is a line connecting them. As can be seen, the model of the agent lattice is much closer to the real evolutionary mechanism in nature than the model of the population in traditional EAs.

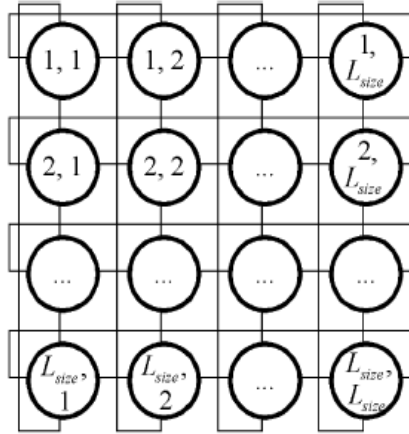


Fig. 1. Agent lattice

2.3 Behaviors of Agents

The goal of an algorithm for T-colorings is to find solutions by a computational cost as low as possible. So the computational cost can be considered as the resources of the environment in which all agents live^[10]. Since the resources are limited and the behaviors of the agents are driven by their purposes, an agent will compete with others to gain more resources. On the basis of this, two behaviors are designed for agents to realize their purposes, that is, the competitive behavior and the self-learning behavior.

A. Competitive behavior

In this behavior, the energy of a solution agent is compared with those of neighbors. The agent can survive if the energy is maximum; otherwise the agent must die, and

the child of the one with maximum energy among the neighbors will take up the lattice-point.

Suppose that the competitive behavior is performed on the agent located at (i, j) , which is labeled as $L_{i,j}$, and $Max_{i,j}$ is the agent with maximum energy among the neighbors of $L_{i,j}$, that is, $Max_{i,j} \in Neighbors_{i,j}$ and $\forall agent \in Neighbors_{i,j}, agent(E) < Max_{i,j}(E)$. If $L_{i,j}(E) < Max_{i,j}(E)$, then $Max_{i,j}$ generates a child agent, namely $Child_{i,j}$, to replace $L_{i,j}$, and the method is shown in Algorithm 1; otherwise, $L_{i,j}$ is left untouched.

Algorithm 1. Competitive behavior

Input: $Max_{i,j}$: $Max_{i,j}(P) := \langle m_1, m_2, \dots, m_N \rangle$;
 L_{size} : The scale of the agent lattice;
 P_c : The parameter used in the competitive behavior;
Output: $Child_{i,j}$: $Child_{i,j}(P) := \langle c_1, c_2, \dots, c_N \rangle$;

$Swap(x, y)$ exchanges the value of x and y . $rand(0, 1)$ is a uniform random number between 0 and 1. $Random(N, i)$ is a random integer among 1, 2, ..., N and is not equal to i . Here N is the number of vertexes of the graph G , namely the different scale of the problems.

begin

$Child_{i,j}(P) := Max_{i,j}(P)$;

$i := 1$;

$Pos := n + 1$;

repeat

if ($rand(0, 1) < P_c$) **then**

begin

$l := Random(N, i)$;

$Swap(c_i, c_l)$;

end;

$i := i + 1$;

until ($i > N$) ;

$Child_{i,j}(SL) := True$;

end.

B. Self-learning behavior

Agents have knowledge relating to the problem that they are designed to solve. Therefore, we designed the self-learning behavior for agents by making use of local search techniques. In this behavior, agent used its knowledge to enhance the energy in order to boost up its competitive ability. After the behavior is performed on an agent, the probability of increasing its energy by this behavior for the second time gets very low, thus $L_{i,j}(SL)$ is set to *False* at the last step.

2.4 Implementation of MAEA-T-coloring

At each generation, the competitive behavior is performed on each agent first. As a result, the agents with low energy are cleaned out from the agent lattice so that there is more space developed for the agents with higher energy, and then the self-learning behavior is performed according to the self-learning flag of the agent. In order to reduce the computational cost, this behavior is only performed on the best agent in the current agent lattice. The whole process is performed iteratively until the maximum computational cost is reached.

Algorithm 2. Self-learning behavior

Input: $agent_{best}$: The agent with the highest energy in current generation, and
 $agent_{best}(P) := \langle p_1, p_2, \dots, p_N \rangle$;

Output: $agent_{learn}$: The result after self-learning behavior, and $agent_{learn}(P) := \langle a_1, a_2, \dots, a_N \rangle$;

begin
 $agent_{learn}(P) := agent_{best}(P)$;
repeat
 $Repeat := False$; $k := 1$; $Iteration := 1$;
while ($k \leq N$) **do**
begin
 $Energy_{best} := agent_{best}(E)$;
 $l := Random(N, i)$;
 $Swap(a_i, a_l)$; $Energy_{learn} := agent_{learn}(E)$;
if ($Energy_{best} \geq Energy_{learn}$) **then** $Swap(a_i, a_l)$
else begin $Repeat := True$; $k := k + 1$; **end** ;
if ($Iteration < N - 1$) **then** $Iteration := Iteration + 1$
else begin $Iteration := 1$; $k := k + 1$; **end** ;
end ;
until ($Repeat = True$) ;
 $L_{i,j}(SL) := False$;

end.

3 Experimental Results

3.1 Benchmarks

As no benchmark is available for T-colorings, we follow the method referred in [3] to develop a random instance generator for restricted T-colorings. According to this way, each instance is defined by three parameters:

- (a) N : the number of vertexes;
- (b) $d \in [0, 1]$: the edge density;
- (c) Sep : the largest color separation.

To generate such an instance, we first build a graph of N vertexes with $d(N(N-1))/2$ edges uniformly distributed on these vertexes. Then, a uniform random value from $[1, Sep]$ is assigned to each edge. So we build 15 random instances of restricted T-coloring denoted by $N.d.Tcol$ with the following possible values: $N \in \{30, 100, 300, 500, 1000\}$, $d \in \{0.1, 0.5, 0.9\}$, $Sep=5$ (see Table 1). In this table, E stands for the number of edges of different graphs. All these instances can be divided into 5 suites according to the different number of vertexes.

Algorithm 3. MAEA-T-coloring

Input: $Evaluation_{max}$: The maximum number of evaluations for the energy;
 V_{size} : The number of vertexes of the graph G ;
 L_{size} : The scale of the agent lattice;
 P_c : The parameter used in the competitive behavior;
Output: A solution or an approximate solution for the T-coloring problem under consideration;

L^t is the agent lattice in the t th generation. $agent_{best}^t$ is the best agent in $L^0, L^1, \dots, L^t$, and $agent_{tbest}^t$ is the best agent in L^t .

begin

for $i := 1$ **to** L_{size} **do**

for $j := 1$ **to** L_{size} **do**

begin

 Generate a permutation randomly and assign it to

$L_{i,j}^0(P)$;

 Compute $L_{i,j}^0(E)$; $L_{i,j}^0(SL) := True$;

end;

$Evaluations := L_{size} \times L_{size}$; Update $agent_{best}^0$; $t := 0$;

repeat

for $i := 1$ **to** L_{size} **do**

for $j := 1$ **to** L_{size} **do**

begin

if ($L_{i,j}^t$ wins in the competitive behavior) **then**
 $L_{i,j}^{t+1} := L_{i,j}^t$

else $L_{i,j}^{t+1} := Child_{ij}$ (generated according to Algorithm 1);

 Compute $L_{i,j}^{t+1}(E)$;

```

    Evaluations := Evaluations + 1 ;
end
Update  $agent_{(t+1)best}^{t+1}$  ;
if ( $agent_{(t+1)best}^{t+1}(SL) = True$ ) then
    Perform the self-learning behavior on  $agent_{(t+1)best}^{t+1}$  ;
if ( $agent_{(t+1)best}^{t+1}(E) < agent_{best}^t(E)$ ) then
begin
     $agent_{best}^{t+1} := agent_{best}^t$  ;
     $agent_{worst}^{t+1} := agent_{best}^t$  ( $agent_{worst}^{t+1}$  is the agent with the lowest energy in  $L^{t+1}$ );
end
else  $agent_{best}^{t+1} := agent_{(t+1)best}^t$  ;
     $t := t + 1$  ;
until ( $Evaluations \geq Evaluation_{max}$ ) ;

end.
```

Table 1. The instances used in the experiment

instances	N	d	E
30.1.Tcol	30	0.1	43
30.5.Tcol	30	0.5	217
30.9.Tcol	30	0.9	391
100.1.Tcol	100	0.1	495
100.5.Tcol	100	0.5	2475
100.9.Tcol	100	0.9	4455
300.1.Tcol	300	0.1	4485
300.5.Tcol	300	0.5	22425
300.9.Tcol	300	0.9	40365
500.1.Tcol	500	0.1	12475
500.5.Tcol	500	0.5	62375
500.9.Tcol	500	0.9	112275
1000.1.Tcol	1000	0.1	49950
1000.5.Tcol	1000	0.5	249750
1000.9.Tcol	1000	0.9	449550

3.2 Experimental Results

In this section, the performance of MAEA-T-coloring is tested with the instances we generated in Table 1. Table 2 shows the outcomes of MAEA-T-coloring and the classical algorithm *Dsatur* comparatively.

We used two criteria to evaluate the algorithm: sp_{best} and sp_{ave} , which respectively stands for the minimum span we could find and the average value of the minimum span we found in all the runs. The smaller the estimate value is, the better the solution is. The experimental results in Table 2 show that MAEA-T-coloring largely outperforms the *Dsatur* algorithm on all the random instances tested. The computing time necessary to get good T-coloring is high for those large and dense graphs.

Table 2. Comparison between *Dsatur* and MAEA-T-coloring

instances	<i>Dsatur</i>		MAEA-T-coloring	
	sp_{best}	sp_{ave}	sp_{best}	sp_{ave}
30.1. <i>Tcol</i>	9	9.0	8	8.0
30.5. <i>Tcol</i>	23	25.2	19	19.0
30.9. <i>Tcol</i>	40	44.9	34	35.6
100.1. <i>Tcol</i>	23	23.0	17	17.8
100.5. <i>Tcol</i>	63	66.1	58	59.0
100.9. <i>Tcol</i>	124	125.0	112	114.1
300.1. <i>Tcol</i>	40	43.8	35	35.6
300.5. <i>Tcol</i>	159	161.3	156	157.8
300.9. <i>Tcol</i>	304	318.9	292	294.5
500.1. <i>Tcol</i>	61	62.5	55	56.6
500.5. <i>Tcol</i>	240	247.5	231	232.3
500.9. <i>Tcol</i>	483	493.1	473	474.1
1000.1. <i>Tcol</i>	104	105.1	92	93.7
1000.5. <i>Tcol</i>	436	441.4	426	430.3
1000.9. <i>Tcol</i>	896	904.2	879	886.7

4 Conclusion

In this paper, multiagent systems and evolutionary algorithms are combined to form a new algorithm to solve T-coloring problem. Based on the characteristic and encoding method of T-colorings, we designed agent, agent environment and agent behaviors for the algorithm. In the experiment, MAEA-T-coloring obtains good performance on the 15 test problems we generated, which following the same principles defined for generating random graphs for graph coloring. Comparing with the classical algorithm *Dsatur*, experimental results show that MAEA-T-coloring is of an encouraging algorithm in solving large-scale T-coloring problems. It is obvious to see that the combination of multiagent systems and EAs is of high potential in solving complex and ill-defined problems.

References

1. Costa, D.: On the use of some known methods for T-colorings of graphs. *Annals of Operations Research* 41, 343–358 (1993)
2. Dorne, R., Hao, J.-K.: Tabu search for graph coloring, T-colorings and set T-colorings. In: *Meta-heuristics 1998, Theory and Applications*, pp. 33–47. Kluwer Academic Publishers, Boston (1998)
3. Riihijärvi, J., Petrova, M., Mähönen, P.: Frequency allocation for WLANs using graph coloring techniques. In: *WONS*, pp. 216–222 (2005)
4. Hurley, S., Smith, D.H.: Bounds for the frequency assignment problem. *Discrete Mathematics* (167–168), 571–582 (1997)
5. Janczewski, R., Kubale, M., et al.: The T-DSATUR algorithm: An interesting generalization of the DSATUR algorithm. In: *International conference on advanced computer systems* (5), pp. 288–292 (1998)
6. Russell, S.J., Norvig, P.: *A modern approach. Artificial Intelligence*. Prentice-Hall, Englewood Cliffs (1995)
7. Hale, W.K.: Frequency Assignment: Theory and Applications. *IEEE Transactions on Vehicular Technology* 68(12), 1497–1514 (1980)
8. Liu, J., Zhong, W., Jiao, L.: A multiagent evolutionary algorithm for constraint satisfaction problems. *IEEE Trans. Syst., Man, and Cybern. B* 36(1), 54–73 (2006)
9. Zhong, W., Liu, J., Xue, M., Jiao, L.: A multiagent genetic algorithm for global numerical optimization. *IEEE Trans. Syst., Man, and Cybern. B* 34(2), 1128–1141 (2004)
10. Liu, J., Zhong, W., Jiao, L.: Job-Shop Scheduling Based on Multiagent Evolutionary Algorithm. In: Wang, L., Chen, K., Ong, Y.S. (eds.) *ICNC 2005. LNCS*, vol. 3612, pp. 925–933. Springer, Heidelberg (2005)

Non-photorealistic Rendering Using Genetic Programming

Perry Barile, Vic Ciesielski, and Karen Trist

School of Computer Science and Information Technology
RMIT University, Melbourne, 3000, VIC, Australia
{bpasqual,vc}@cs.rmit.edu.au,
karen.trist@rmit.edu.au

Abstract. We take a novel approach to Non-Photorealistic Rendering by adapting genetic programming in combination with computer graphics drawing techniques. As a GP tree is evaluated, upon encountering certain nodes referred to as “Draw” nodes, information contained within such nodes are sent to one of three virtual canvasses and a mark is deposited on the canvas. For two of the canvasses the user is able to define custom brushes to be applied to the canvas. Drawing functions are supplied with little localised information regarding the target image. Based on this local data, the drawing functions are enabled to apply contextualized information to the canvas. The obtained results include a “Shroud of Turin” effect, a “Decal” effect and a “Starburst” effect.

Keywords: genetic programming, non-photorealistic rendering, evolutionary computation.

1 Introduction

The challenge of producing photorealistic imagery has been a long term goal within the field of computer graphics[1,2]. On the other hand, non-photorealistic rendering (NPR) aims at producing images derived from disciplines such as painting, drawing, sketching, illustration and animation. The goal of NPR research in computer science is to develop algorithms and methodologies such that a computer program can emulate the work done traditionally by hand.

Our aim is to explore an evolutionary approach to non-photorealistic rendering, specifically through genetic programming. There has been a great deal of research into NPR in recent years, but few researchers have applied genetic programming to the problem. We have developed a painting program that generates trees composed of drawing commands that are applied to a canvas in order to emulate a specified target image. Each drawing command, represented as a node in the tree, deposits ink on a canvas according to local parameters defined in the node. We have developed some tree representations that utilise properties of the target image and some tree representations that do not.

We have identified that there is little work within the research community on NPR rendering using genetic programming. We provide a set of methodologies

that govern the production of non-photorealistic images using genetic programming. Moreover, we have identified four characteristics of our approach that set it apart from previous NPR work.

1. **GP approach.** While there have been some efforts in NPR research using evolutionary algorithms [3,4,5], there is very little work being conducted on NPR using genetic programming.
2. **Automatic evaluation.** Many approaches to NPR are interactive and require human input in order to determine the subjective appeal of output images. Our approach differs in the fact that the evolutionary process is not guided by human judgment, but rather by a fitness function. We are not proposing a fitness function for evaluating aesthetic appeal, but our approach allows for the emergence of computer-generated aesthetic qualities.
3. **Animated outcome.** A key feature of the evolutionary process is that it is generally interesting to observe. We believe that the journey may be just as, or more interesting, than the destination. As a consequence of this, our final output is not simply a *best* image, but an animated sequence of best images over the entire evolutionary process.

We have identified several sources in which preconceived artistic styles were emulated [2,5,6,7,8,9,10,11,12]. In other works, a novel form of rendering was discovered [1,3,4,13]. Our aim is to apply genetic programming to target images in order to discover novel program representations of such images. We hypothesize that by implementing our own approach to genetic programming, we can discover novel forms of non-photorealistic rendering. Additionally we seek to investigate the effects of applying programmatic methodologies and domain knowledge in order to promote varieties of rendering styles and to promote convergence. Our research questions are:

1. Can we discover novel forms of non-photorealistic rendering?
2. What methodologies can we employ in order to facilitate different drawing styles?
3. What types of domain knowledge can be used in order to promote faster convergence?

2 Related Work and Methods

In this section we first discuss work by other researchers, then we discuss the two primary methods of fitness evaluation.

2.1 Related Work

In [9], Haeberli suggests several techniques designed to facilitate informed heuristics in order to promote various visual effects. In [3], Lioret provides a brief survey of rendering techniques involving cellular automata, Lindenmayer systems, fractals, morphogenesis, neural networks and genetic algorithms.

In [6], Baxter et al developed a tool which models natural media in order to simulate the flow and drying of paint. Another approach to simulating natural media is [7], in which East Asian ink and watercolour techniques are simulated.

In [8], Gooch et al take an interesting approach in which they take a photograph and apply transformations based on brightness and luminance in order to produce line art portraits. In [12], Wen et al, take input images and apply a loose segmentation algorithm to them. Drawing on the canvas is achieved by consulting a database of colours and applying a set of drawing rules defined by experts. A form of line art, with colored strokes and filled areas, is achieved.

In [10], Hertzmann takes a target image and applies a transformation or effects filter to it. A learning algorithm is employed to “observe” the transformation. The algorithm is then applied to a different target image in order to reproduce the effects of the learned filter. In [13], Hertzmann defines sets of *energy functions* that allow definitions of desires for painting and their relative importance. These functions allow different styles of painting. The system consists of applying strokes using these functions and through various means of *relaxing* the brush strokes, various effects are applied to input images.

In [1], Semet et al employ a distributed agent approach in which a colony of “ants” populates the canvas and deposit marks on the canvas according to local information, such as edge-detected gradients.

In [14], Wijesinghe et al generate an animated mosaic that converges towards a target image.

2.2 Fitness Evaluation

Many NPR approaches require human involvement to interactively judge the aesthetic appeal of images. There are weaknesses in this approach. For instance, humans can only stay focussed for relatively short periods. However some NPR applications are specifically designed in order to function interactively with users. The IMPasTo project[6] aims at allowing users to explore the digital painting process. In the work of Semet et al.[1], the user drives the gradual development of characteristics of ant colonies in order to emulate the sketching process of starting out with rough strokes and then gradually refining the strokes.

In contrast to the interactive process, many approaches employ an automatic software function to make judgements on the appeal of images[15]. Various techniques have been used to compute the fitness of an image. Most approaches (such as ours) adopt the simple heuristic of comparing an output image pixel-by-pixel to the target image. There are other approaches. In [16], Machado et al. base fitness on image complexity and uses a formula based on quad-trees in order to measure complexity. In [17], Ross et al. utilise Ralph’s bell curve model of aesthetics, based on empirical analysis of fine art, in order to evolve aesthetically pleasing images.

3 Our Approach

This section details our modified GP approach, then describes our specialized draw functions and GP configuration.

3.1 Modified GP Approach

Our representation uses GP techniques in a non-conventional manner. There are two type of function nodes: *Prog* nodes and *Draw* nodes. No values are passed up the tree. Evaluation of a GP tree results in a sequence of Draw strokes on the canvas. The GP formalism provides a convenient representation for a variable number of brush strokes and the order in which they are executed. See figure 1.

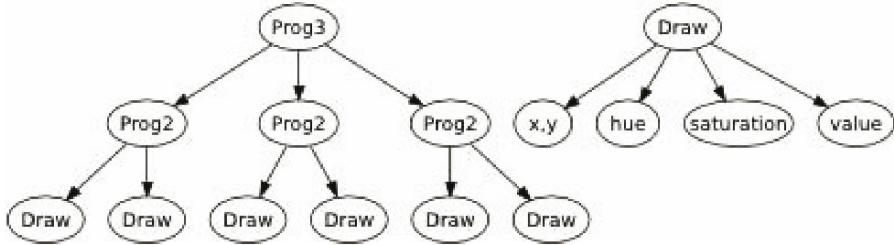


Fig. 1. This figure shows the relationship between Prog nodes and Draw nodes. The Draw node shown on the right of the figure would be used for issuing a drawing command at a position on the canvas, employing one input for the position and three input for each channel of an HSV colour.

We employ **Prog2** and **Prog3** nodes, which accept 2 or 3 inputs respectively. **Draw** nodes can only have terminals (random float nodes) as input. Terminals are float variables which are initialised with a value between 0.0 and 1.0 using uniform probability distribution. Prog nodes serve merely to pad the size of the tree and to spread out draw nodes. When the tree is evaluated, anytime that a Draw node is encountered, a brush stroke is made on the canvas using the information contained in each Draw node. Each canvas has its own interpretation of the information sent it by a Draw node.

We have created three canvasses used to perform rendering. Only one canvas is used at a time. We call the canvasses: “Shroud of Turin”, “Decal” and “Starburst”. The Draw node depicted in figure 1 pertains to the Starburst canvas and the Decal canvas. In the case of the Shroud of Turin canvas, the inputs to the Draw node are: x/y , *stroke length*, *stroke angle*, *grayscale stroke color*. Again, each canvas has its own interpretation of the information sent it by a Draw node. The Draw nodes themselves are generic. The input label “ x,y ” may be confusing and suggests that two nodes are in fact necessary, but this is not the case. Each canvas is represented as a 2D array of pixels. It is a trivial matter to represent a 2D array as a 1D pointer and determine the x and y offset of a pixel using the width of the canvas.

3.2 Draw Functions

During the course of our experiments we identified a range of input values for Draw nodes, according to a desired pictorial outcome. In addition to this, we identified some choices to be made when implementing Draw functions.

Over-writing Previously Drawn Pixels: What to do when about to draw a pixel that has already been drawn? Three distinct choices that were made:

1. Replace the existing pixel
2. Blend the existing pixel with the new pixel by computing their average value
3. Do nothing (preserve the existing pixel)

Incorporating Localized Image Data in Drawing Calculations: The target image is pre-processed by converting it from the RGB color space to the HSV color space. Gradient maps and an importance map are prepared for the target image. Examples of these maps are shown in figure 2.

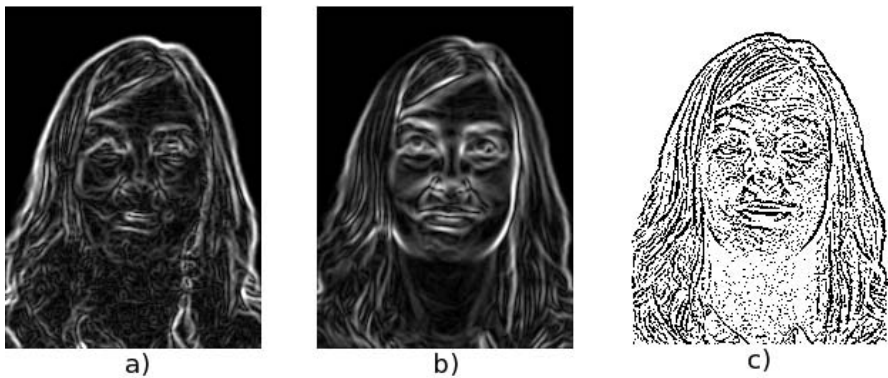


Fig. 2. a) A saturation gradient map. b) A value gradient map. c) An importance map.

Gradient maps: Edge detection using a 3x3 Sobel filter is applied to both the *saturation* channel and the *value* channel of the target image. This produces a *saturation edge map* and a *value edge map*.

Importance map: A user can specify parts of an image considered important. Whenever a pixel is about to be drawn, six pieces of pixel data are considered: the pixel data of the pixel about to be deposited (Pb), the pixel on the canvas about to be drawn on (Pc) and the corresponding pixel of the target image (Pt); the corresponding pixel in the saturation map (Ps), the corresponding pixel in the value map (Pv) and the corresponding pixel in the importance map (Pm). A threshold value (T) of 20 in a range of 0 to 255 is set. If Ps is above T , saturation matching is performed. If Pv is above T , value matching is performed. If Pm is above T , both saturation and value matching is performed. Hue matching is always performed.

Hue / Saturation / Value Matching. For each channel, whether it be hue, saturation or value, the following matching function is used:

1. Compare Pb to Pt .
2. Compare Pc to Pt .
3. For each channel, if and only if the channel value in Pb is closer to Pt than Pc , draw that channel value.

3.3 GP Configuration

The configuration for genetic programming is shown in table 1. We use Strongly Typed GP.

Table 1. Genetic Programming Configuration

Parameter	Value
Population Size	10
Crossover Rate	40%
Mutation Rate	50%
Elitism Rate	10%
Max. Generations	25000
Selection	Roulette Wheel
Initialization	Decimation from initial population of 100
Replacement Strategy	Generational replacement
Fitness	Pixel-by-pixel comparison
Termination	0.05 difference or max. generations reached
Functions	Prog2, Prog3, Draw
Terminal	
<i>frand</i>	random floating point number between 0.0 and 1.0

4 Experiments and Results

During exploration of the creative range of our system, we identified 3 pictorial styles that can be generated by applying particular image pre-processing steps and drawing functions. In all cases the inputs to Draw functions are *frand* terminals. Runs typically take several hours to complete. We detail these effects below.¹

4.1 Shroud of Turin Effect

Image Pre-Processing: None.

Channel Matching: None.

Average Tree Size: 12186

Number of Draw Nodes: 1945

¹ The printed format does not convey the visual richness of some of our results. Additionally our results are created as MPEG files. To view our results in full, please go to <http://www.cs.rmit.edu.au/~vc/evolved-art/npr-seal/>

Draw Function: This effect is so named because evolutionary sequences typically consist of a “ghostly” image gradually emerging into a “corporeal” form. The effect is best applied to grayscale targets. The Draw function takes 4 inputs and draws a straight line. The inputs define *starting coordinate*, *line length*, *line angle* and *line grayscale color*. See figure 3. As stated previously, the *x* and *y* coordinates of the starting coordinate can be computed using only one input value.

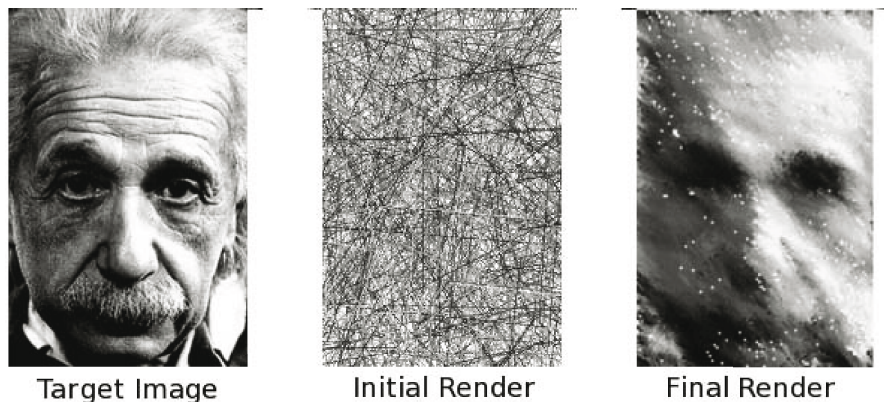


Fig. 3. Shroud Of Turin Effect

4.2 Starbursts

Image Pre-Processing: Saturation and Value Maps generated, Importance map specified by the user.

Channel Matching: Hue, saturation, value channels.

Average Tree Size: 1955

Number of Draw Nodes: 349

Draw Function: The effect is defined by an internal (programmer-defined) 2D array of booleans. The array consists of a set of “spokes” arranged around a central hub. The radius of the hub is programmer-defined. We set the radius to be computed as $1/20$ th of the width or height of the input image, whichever is smaller. The spokes are defined as true values. The entire array is processed. Any cell values in which the value is false are not drawn, hence achieving a transparent effect around the spokes. The Draw function accepts 4 inputs which define *the central pixel coordinate of each Starburst*, and the *hue*, *saturation* and *value* channels of the color to be applied by the Starburst. Each time a pixel is drawn, the color of the existing pixel is blended with the color of the current starburst. See figure 4.

4.3 Decals

Image Pre-Processing: Saturation and Value Maps generated, Importance map specified by the user.

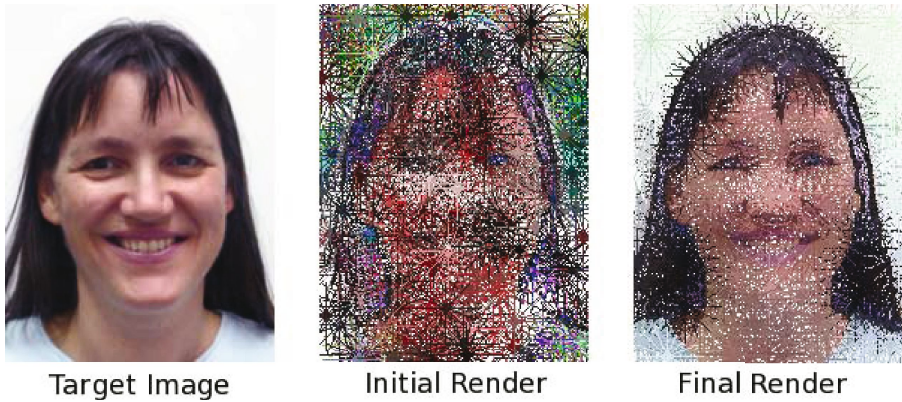


Fig. 4. Starburst Effect

Channel Matching: Hue, saturation, value channels.

Average Tree Size: 1518

Number of Draw Nodes: 272

Draw Function: With this effect, an external image (the decal), is specified by the user. The decal consists of grayscale pixels. Pixels values in the Decal image that tend toward black are drawn with greater opacity, while values that tend toward white are drawn with lower opacity. The Draw function accepts 4 inputs which define *the central pixel coordinate of each Decal*, and the *hue, saturation and value* channels of the color to be applied by the Decal. When a pixel is drawn, the color of existing pixel is blended with the color of the Decal pixel, subject to the opacity of the Decal pixel. See figure 5.

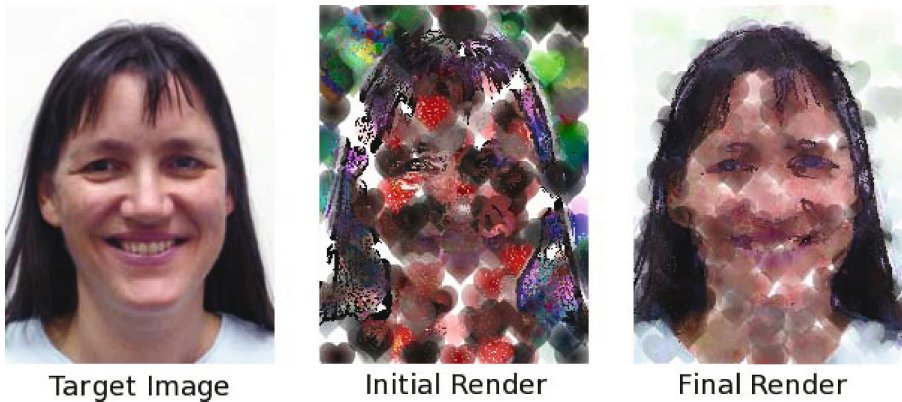


Fig. 5. Decals Effect

5 Conclusions and Future Work

To answer our research questions:

Can we discover new forms of non-photorealistic rendering?

- The “Shroud of Turin” effect was discovered as a result of applying a simplistic drawing function that draws straight strokes, utilising no local image data such as areas of importance or edge maps.
- The “Starburst” and “Decal” effects employ a “paint blotch” rendering technique. The Starburst effect employs color compositing using full opacity, while the Decal effect utilises transparency while compositing.

What methodologies can we employ in order to facilitate different drawing styles? To achieving stylised looks, we focussed on exactly what happens when a stroke is deposited on the canvas. We explored several stroke-based techniques, but failed to achieve any effects that evoked genuine excitement from our artistic collaborators. Switching from a stroke based approach to the decal approach elicited interest from our advisors. We found that hue/saturation/value channel matching had to be employed in order to promote convergence.

What types of domain knowledge can be used in order to promote faster convergence? Applying channel data matching when compositing pixels speeds convergence considerably. Allowing the user to specify important areas on the target image allows the program to more accurately render important details within the image. Applying edge detection provides the program with important information regarding changes in contrast and tonality, speeding convergence.

We have identified several issues that we intend to investigate in future work. We intend to investigate methods of optimizing our generated GP structures. Furthermore we intend to investigate alternate evolutionary forms, such as linear genetic programming and grammatical evolution, in order to produce string representations of our GP trees. We also intend to investigate methodologies for evolving line drawings. Future work will also focus on incorporating aesthetic sense into the fitness function.

References

1. Semet, Y., O'Reilly, U.-M., Durand, F.: An Interactive Artificial Ant Approach To Non-Photorealistic Rendering. In: Deb, K., et al. (eds.) GECCO 2004. LNCS, vol. 3103. Springer, Heidelberg (2004)
2. Strothotte, T., Schlechtweg, S.: Non-Photorealistic Computer Graphics: Modeling, Rendering and Animation. Morgan Kaufmann, San Francisco (2002)
3. Lioret, A.: Being Paintings. In: ACM SIGGRAPH 2005 Electronic Art and Animation Catalog (2005)
4. Sims, K.: Artificial evolution for computer graphics. In: SIGGRAPH: Proceedings Of The 18th Annual Conference On Computer Graphics And Interactive Techniques (1991)

5. Terzopoulos, D.: Artificial Life For Computer Graphics. *Commun. ACM* 42(8), 32 (1999)
6. Baxter, W., Wendt, J., Lin, M.C.: IMPasTo: A Realistic, Interactive Model For Paint. In: *NPAR: Proceedings Of The 3rd International Symposium On Non-Photorealistic Animation And Rendering* (2004)
7. Chu, N.S.-H., Tai, C.-T.: MoXi: Real-time Ink Dispersion In Absorbent Paper. In: *ACM SIGGRAPH* (2005)
8. Gooch, B., Reinhard, E., Gooch, A.: Human Facial Illustrations: Creation And Psychophysical Evaluation. *ACM Transaction On Graphics* (2004)
9. Haeberli, P.: Paint By Numbers: Abstract Image Representations. *Computer Graphics* 24(4), 207–214 (1990)
10. Hertzmann, A.: Image Analogies. In: *SIGGRAPH: Proceedings Of The 28th Annual Conference On Computer Graphics And Interactive Techniques* (2001)
11. Hertzmann, A.: A Survey Of Stroke-Based Rendering. *Computer Graphics and Applications* (2003)
12. Wen, F., Luan, Q., Liang, L., Xu, Y.-Q., Shum, H.-Y.: Color Sketch Generation. In: *NPAR: Proceedings Of The 4th International Symposium On Non-Photorealistic Animation And Rendering*, pp. 47–54 (2006)
13. Hertzmann, A.: Paint By Relaxation. In: *Proceedings of Computer Graphics International*, pp. 27–54 (2001)
14. Wijesinghe, G., Mat Sah, S.B., Ciesielski, V.: Grid vs. Arbitrary Placement of Tiles for Generating Animated Photomosaics. In: *2008 World Congress on Computational Intelligence* (2008)
15. McCormack, J.: New Challenges For Evolutionary Music And Art. *SIGEVolution* 1, 5–11 (2006)
16. Machado, P., Cardoso, A.: All The Truth About NEvaR. *Applied Intelligence* 16(2), 101–118 (2002)
17. Ross, B.J., Ralph, W., Zong, H.: Evolutionary Image Synthesis Using a Model of Aesthetics. In: *Proc. 2006 IEEE Congress on Evolutionary Computation* (2006)

Use of Local Ranking in Cellular Genetic Algorithms with Two Neighborhood Structures

Hisao Ishibuchi, Noritaka Tsukamoto, and Yusuke Nojima

Department of Computer Science and Intelligent Systems, Graduate School of Engineering,
Osaka Prefecture University, 1-1 Gakuen-cho, Naka-ku, Sakai, Osaka 599-8531, Japan

hisaoi@cs.osakafu-u.ac.jp, nori@ci.cs.osakafu-u.ac.jp,
nojima@cs.osakafu-u.ac.jp

http://www.ie.osakafu-u.ac.jp/~hisaoi/ci_lab_e

Abstract. In our former study (Ishibuchi et al. 2006), we proposed the use of two neighborhood structures in a cellular genetic algorithm. One is for local selection where a pair of parents is selected from neighboring cells for mating. This neighborhood structure has been usually used in standard cellular algorithms. The other is for local competition, which is used to define local elitism and local ranking. We have already examined the effect of local elitism on the performance of our cellular genetic algorithm (Ishibuchi et al. 2008). In this paper, we examine the effect of using local ranking as the fitness of each individual. First we explain our cellular genetic algorithm with the two neighborhood structures. Then we examine its two variants with/without local ranking. In one variant, the local ranking of an individual among its neighbors is used as its fitness. Such a fitness redefinition scheme can be viewed as a kind of noise in parent selection. The other variant uses the original fitness value (instead of its local ranking). Through computational experiments, we demonstrate that the use of the local ranking improves the ability to escape from local optima.

1 Introduction

Since early studies in the late 1980s [1], [2] and the early 1990s [3], [4], cellular algorithms have been an active research area in the field of evolutionary computation (e.g., see [5]–[8]). In cellular algorithms, each individual is spatially fixed in a cell of a lattice (typically a two-dimensional grid-world). A new offspring in a cell is generated from individuals in its neighboring cells. The main characteristic of cellular algorithms is the use of local selection, which is based on a neighborhood structure. A single neighborhood structure has been usually used in standard cellular algorithms.

There are, however, many cases where biological evolution is based on two neighborhood structures: One is for local competition and the other is for local selection. For example, most plants have these two neighborhood structures. That is, neighboring plants fight with each other for water and sunlight in the competition neighborhood, which is much smaller than the selection neighborhood where they can disperse their pollen. Another example is territorial animals. The competition neighborhood (i.e., territory) of a territorial animal is much smaller than its selection neighborhood for mating. Evolution of altruism in a two-dimensional grid-world with the two

neighborhood structures was actively studied under the name of structured demes in the late 1970s [9], [10], [11]. Recently the two neighborhood structures have been used to analyze the evolution of cooperative behavior in spatial prisoner's dilemma games (e.g., [12], [13]). It was shown in [13] that cooperative behavior was evolved even in the case of random pairing in game playing when a small competition neighborhood was used together with a large selection neighborhood.

In our former study [14], we proposed a cellular genetic algorithm with the two neighborhood structures for function optimization problems. Good results were obtained from the combination of a small competition neighborhood and a large selection neighborhood. In our cellular genetic algorithm, the selection neighborhood was used for local selection as in standard cellular algorithms while the competition neighborhood was used for defining local elitism and local ranking. Locally best individuals were handled as elite in the local elitism. The local ranking of an individual among its neighbors was used as the fitness in parent selection. Since the competition neighborhood was used for both the local elitism and the local ranking, it was difficult in [14] to explain the effect of introducing the competition neighborhood. So we examined the pure effect of the local elitism in [15] where the local ranking was not used in our cellular genetic algorithm. Better results were reported for some test problems by the use of the local elitism than standard cellular algorithms where the "replace-if-better" scheme was applied to all cells. In the local elitism, such a hill-climbing replacement rule was applied only to locally best individuals.

In this paper, we try to examine the pure effect of the local ranking on the performance of our cellular genetic algorithm. First we explain our cellular genetic algorithm with the two neighborhood structures [14] in Section 2. We also explain three implementation schemes of elitism (i.e., local, global and cell-wise [15]). Then we examine two variants of our cellular genetic algorithm with/without the local ranking through computational experiments in Section 3. In one variant, we use the local ranking of an individual in its competition neighbors as its fitness. In the other variant, we do not use the local ranking. That is, each individual is always evaluated by its original fitness value as in standard cellular algorithms. The two variants are applied to a knapsack problem and a minimization problem of Schwefel function. Experimental results show that the use of the local ranking improves the ability of our cellular genetic algorithm to escape from local optima. Finally Section 4 concludes this paper.

2 Cellular Genetic Algorithms with Two Neighborhood Structures

In this section, we briefly explain our cellular genetic algorithm. Three implementation schemes of elitism (i.e., local, global and cell-wise) are also explained.

2.1 Two Neighborhood Structures: Local Competition and Local Selection

We use a two-dimensional grid-world where a single individual is spatially fixed in each cell. Thus the number of cells is the same as the population size. We assume the torus structure in our two-dimensional grid-world. In Fig. 1, we show some typical examples of neighborhood structures used in the literature. In each plot, open circles are the neighbors of the closed circle individual.

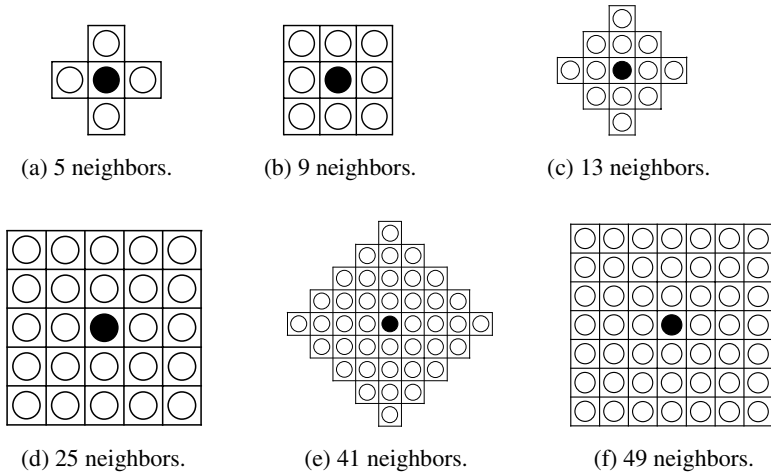


Fig. 1. Six neighborhood structures examined in this paper

As we have already explained, we use the two neighborhood structures in our cellular genetic algorithm: Competition neighborhood and selection neighborhood. We denote the two neighborhood structures of the i th cell as $N_{\text{Compete}}(i)$ and $N_{\text{Select}}(i)$, respectively. Each neighborhood structure shows a set of neighboring cells around the i th cell. The i th cell itself is included in $N_{\text{Compete}}(i)$ and $N_{\text{Select}}(i)$.

We use the six neighborhood structures in Fig. 1 for local competition and local selection. In our computational experiments, all the 6×6 combinations in the size of the two neighborhood structures are examined. The number of cells in a neighborhood structure is referred to as the neighborhood size. For example, the size of the neighborhood structure in Fig. 1 (a) is 5 while it is 49 in Fig. 1 (f).

We use binary tournament selection to select a pair of parents for the i th cell from its selection neighborhood $N_{\text{Select}}(i)$. We examine two variants of our cellular genetic algorithm with/without the local ranking. In one variant (called the rank-based algorithm), we assign a rank to each individual according to the ranking of its fitness value among its neighbors in the competition neighborhood. Let us denote the individual in the i th cell by $\mathbf{x}_i$. In order to assign a rank to $\mathbf{x}_i$, we sort its neighbors (including $\mathbf{x}_i$ itself) in $N_{\text{Compete}}(i)$ in a descending order of their fitness values. When multiple neighbors have the same fitness value, we use a random tiebreak procedure in the sorting. A rank is assigned to $\mathbf{x}_i$ according to its position in the sorted list of its neighbors. Let us explain the rank assignment procedure using a very simple example. In Fig. 2, we show a part of a two-dimensional grid-world. The uppercase letter in the top-left corner of each cell is the label of the cell. For simplicity, this label is also used to denote the individual in the corresponding cell. The fitness value of the current individual in each cell is shown by an integer in the cell. It is assumed that the larger fitness is the better in Fig. 2. The bold lines show the competition neighborhood of the cell F. That is, we use the neighborhood structure in Fig. 1 (a) as the competition neighborhood. In Fig. 2, rank 2 is assigned to the individual F because it has the second best fitness value in its competition neighborhood (i.e., B, E, F, G and J). On the

A 3	B 2	C 8	D 4
E 1	F 4	G 6	H 7
I 2	J 3	K 9	L 8

Fig. 2. Cell labels (uppercase letters) and fitness values (integers)

other hand, rank 4 is assigned to the individual G since its fitness value is the fourth in its competition neighborhood (i.e., C, F, G, H and K). While the individual G has a larger fitness value than F, F has a better rank than G. This simple example clearly shows how local competition in the competition neighborhood changes the strength of each individual for local selection in the rank-based algorithm. In the other variant called the fitness-based algorithm, we use the original fitness value of each individual (e.g., 4 for F and 6 for G).

2.2 Three Implementation Schemes of Elitism: Global, Cell-Wise, and Local

As in our former study [15], we use three implementation schemes of elitism: global, cell-wise and local. It should be noted that the evaluation of each individual in elitism is always based on its fitness value (not its rank). The rank of each individual is used only for parent selection in the rank-based algorithm. It should be also noted that we use the replace-if-better policy only for elite individuals in this paper. Every non-elite individual is always replaced with a newly generated offspring.

Global elitism: A prespecified number of the best individuals in the entire population are handled as elite individuals in the global elitism. The global elitism is independent of the two neighborhood structures. The replace-if-better policy is applied only to the globally best individuals in the global elitism.

Cell-wise elitism: The replace-if-better policy is applied to all cells. That is, all individuals are handled as elite. The cell-wise elitism can be viewed as a special case of the global elitism where the number of elite individuals is the population size.

Local elitism: The implementation of the local elitism depends on the competition neighborhood. When an individual has the highest fitness value among its neighbors in the competition neighborhood, it is handled as elite. That is, the replace-if-better policy is applied only to the locally best individuals in the local elitism.

As in our former study [15], we use three versions of the local elitism, which are different in the handling of the tie situation where the current individual in a cell has the same locally best fitness value as some neighbors in its competition neighborhood. Let K be the number of neighbors (including the current individual in the cell) that have the same locally best fitness value in the competition neighborhood. Version 1 handles the locally best individual as elite only when it is better than all the other neighbors (i.e., only when $K=1$). Version 2 probabilistically determines whether the locally best individual is elite or not in the tie situation. The probability of being elite is specified as $1/K$. Version 3 always handles the locally best individual as elite even when some other neighbors (or all neighbors) have the same locally best fitness value.

3 Performance Evaluation of Our Cellular Genetic Algorithm

In this section, we examine the effect of using the local ranking by comparing the two variants of our cellular genetic algorithm (i.e., the fitness-based and rank-based algorithms) through computational experiments.

3.1 Setting of Computational Experiments

As a test problem, we used a 500-item 0/1 knapsack problem with two constraint conditions. This test problem was generated from the two-objective 500-item problem in Zitzler & Thiele [16] by defining a fitness function as $fitness(\mathbf{x}) = f_1(\mathbf{x}) + f_2(\mathbf{x})$ where $\mathbf{x}$ is a 500-dimensional binary vector, $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$ are the two objectives of the original 2-500 problem in [16]. As another test problem, we used a minimization problem of Schwefel function with ten variables. Each variable of Schwefel function was coded as a binary string of length 10 using gray coding.

We applied our cellular genetic algorithm to these two test problems using the following parameter specifications:

- Grid-world: 11×11 (population size: 121),
- Competition neighborhood: One of the six neighborhood structures in Fig. 1,
- Selection neighborhood: One of the six neighborhood structures in Fig. 1,
- Crossover probability (uniform crossover): 0.8,
- Mutation probability (bit-flip mutation): $1/(\text{string length})$,
- Constraint handling in the knapsack problem: Greedy repair as in [16],
- Stopping condition: 2000 generations.

We examined all the 6×6 combinations of the two neighborhood structures. Four specifications of the number of elite individuals were examined in the global elitism: 1, 6, 12, and 24 (i.e., 1%, 5%, 10% and 20% of the population size 121).

3.2 Experimental Results on the Knapsack Problem

After 2000 generations in each run on the knapsack problem, we calculated the relative error of the obtained solution using the optimal solution. Then the average relative error was calculated over 100 runs for each parameter setting. Experimental results are summarized in Fig. 3 for the fitness-based algorithm and Fig. 4 for the rank-based algorithm. In these figures, the base plane of each plot shows the size of the two neighborhood structures (i.e., the x -axis is the selection neighborhood while the y -axis is the competition neighborhood). Due to the page limitation, we only show the results by the cell-wise elitism and the local elitism Version 1.

We can not observe any clear difference between Fig. 3 and Fig. 4. That is, we can not find any clear effect of using the local ranking in Fig. 4 on the performance of our cellular genetic algorithm. For further examining the effect of the local ranking, we summarized experimental results with the best setting of the two neighborhood structures for each version of elitism in Table 1 for the fitness-based algorithm and in Table 2 for the rank-based algorithm. It should be noted that the competition neighborhood was not used in the fitness-based algorithm with the global and

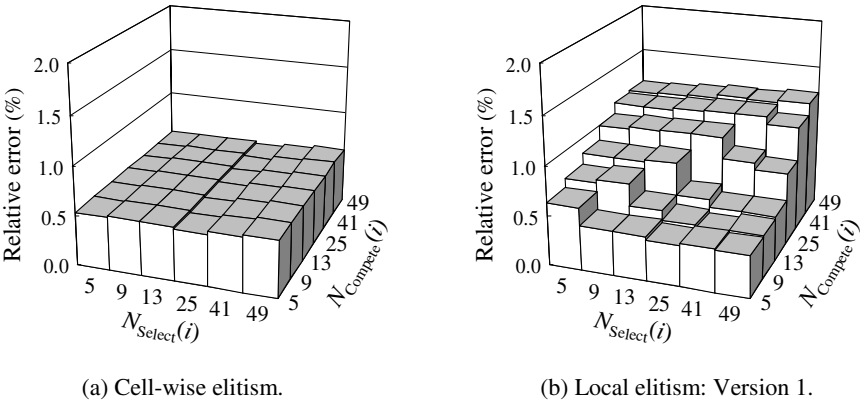


Fig. 3. Average relative errors by the fitness-based algorithm on the knapsack problem

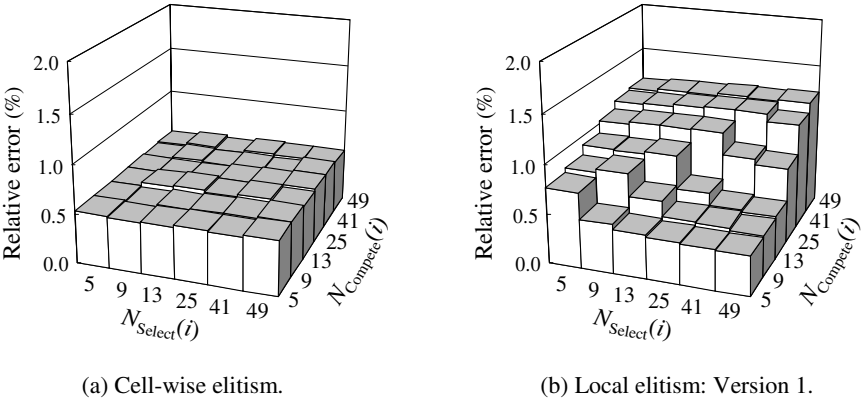


Fig. 4. Average relative errors by the rank-based algorithm on the knapsack problem

Table 1. Best results from each case (Fitness-based algorithm on the knapsack problem)

Elitism	Two neighborhood structures			Single neighborhood	
	Competition	Selection	Relative error	Selection	Relative error
One global elite	-	9	1.019 (0.124)	9	1.019 (0.124)
Six global elites	-	49	0.461 (0.089)	49	0.461 (0.089)
12 global elites	-	41	0.467 (0.099)	41	0.467 (0.099)
24 global elites	-	41	0.479 (0.098)	41	0.479 (0.098)
Cell-wise elitism	-	25	0.524 (0.115)	25	0.524 (0.115)
Local Version 1	9	41	0.410* (0.079)	5	0.636 (0.115)
Local Version 2	9	41	0.436 (0.093)	9	0.500 (0.105)
Local Version 3	5	5	0.543 (0.128)	5	0.543 (0.128)

* The best result in the table.

Table 2. Best results from each case (Rank-based algorithm on the knapsack problem)

Elitism	Two neighborhood structures			Single neighborhood	
	Competition	Selection	Relative error	Selection	Relative error
One global elite	49	9	1.047 (0.132)	121	1.158 (0.115)
Six global elites	49	41	0.467 (0.101)	41	0.469 (0.094)
12 global elites	41	25	0.441 (0.101)	41	0.456 (0.108)
24 global elites	13	41	0.467 (0.106)	41	0.473 (0.104)
Cell-wise elitism	9	9	0.519 (0.105)	9	0.519 (0.105)
Local Version 1	5	49	0.405*(0.109)	5	0.770 (0.105)
Local Version 2	9	41	0.454 (0.117)	5	0.495 (0.104)
Local Version 3	5	13	0.539 (0.120)	5	0.555 (0.125)

* The best result in the table.

cell-wise elitism. For comparison, we also show the corresponding results by the standard cellular algorithm with only the selection neighborhood in the last two columns of Table 1. Moreover, we show the experimental results of the rank-based version of the standard cellular algorithm in the last two columns of Table 2 where the single neighborhood was used not only for the local selection but also for the local ranking and the local elitism. Each real number in parentheses shows the standard deviation of the corresponding average relative error. We can not observe any clear difference between Table 1 and Table 2. That is, the effect of local ranking is still unclear.

3.3 Experimental Results on Schwefel Function

In the same manner as in Table 1 and Table 2, experimental results on the minimization problem of Schwefel function are summarized in Table 3 and Table 4 where the best average value of Schwefel function is shown for each specification of elitism. From the comparison between Table 3 and Table 4, we can see that better results were obtained from the rank-based algorithm in Table 4 than the fitness-based algorithm in Table 3 in many cases (with only a few exceptions). We can also see that the standard deviation was very large in almost all cases in Table 3 and Table 4. Large values of the standard deviation suggest that the evolutionary search was often trapped by different local optima. Moreover, we can see from each table that the use of strong elitism severely degraded the performance of our cellular genetic algorithm and the standard cellular algorithm. This observation suggests that the evolutionary search was often trapped by local optima under strong selection pressure.

Let us further examine the relation between the performance of our cellular genetic algorithm and the strength of elitism. In the global elitism, the number of elite individuals is directly related to the strength of elitism. In Fig. 5, we show how the performance of our cellular genetic algorithm with the global elitism depends on the number of elite individuals. It should be noted that the cell-wise elitism corresponds to the case of 121 elite individuals in the global elitism.

In Fig. 5 (a), the difference between the fitness-based and rank-based algorithms is small. Good results with the relative average error less than 0.55% were obtained by our cellular genetic algorithm (except for the case of a single elite individual) independent of the use of the local ranking. On the contrary, we can observe a clear

Table 3. Best results from each case (Fitness-based algorithm on Schwefel function)

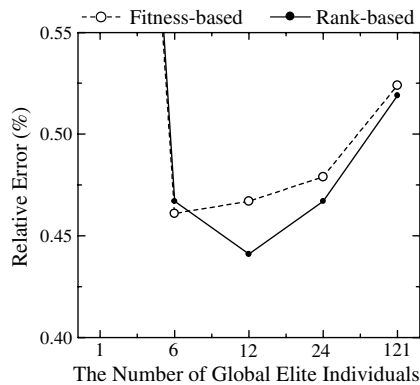
Elitism	Two neighborhood structures			Single neighborhood	
	Competition	Selection	Function value	Selection	Function value
One global elite	-	5	0.014* (0.10)	5	0.014* (0.100)
Six global elites	-	5	9.479 (32.14)	5	9.479 (32.14)
12 global elites	-	5	14.53 (38.48)	5	14.53 (38.48)
24 global elites	-	5	35.54 (68.05)	5	35.54 (68.05)
Cell-wise elitism	-	5	40.28 (71.51)	5	40.28 (71.51)
Local Version 1	41	5	1.529 (11.52)	121	22.89 (48.82)
Local Version 2	41	5	4.761 (23.21)	9	22.51 (49.40)
Local Version 3	13	5	55.68 (84.53)	5	58.05 (82.92)

* The best result in the table.

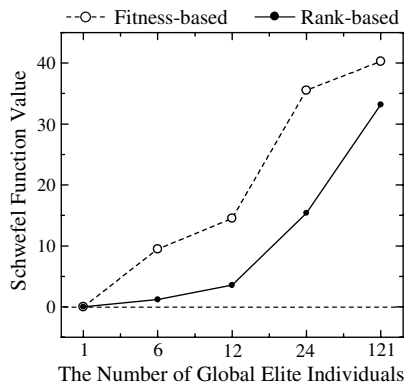
Table 4. Best results from each case (Rank-based algorithm on Schwefel function)

Elitism	Two neighborhood structures			Single neighborhood	
	Competition	Selection	Function value	Selection	Function value
One global elite	13	5	0.001* (0.00)	5	2.371 (16.59)
Six global elites	5	9	1.186 (11.79)	5	1.655 (12.63)
12 global elites	5	5	3.555 (20.21)	5	3.555 (20.21)
24 global elites	9	5	15.40 (43.22)	5	16.59 (41.11)
Cell-wise elitism	13	5	33.17 (60.59)	5	41.46 (71.82)
Local Version 1	49	5	1.984 (12.52)	5	14.22 (38.50)
Local Version 2	41	5	1.216 (11.78)	5	14.22 (41.98)
Local Version 3	5	5	36.73 (66.37)	5	36.73 (66.37)

* The best result in the table.



(a) Knapsack problem.



(b) Schwefel function.

Fig. 5. Relation between the performance of our algorithm and the strength of elitism

positive effect of using the local ranking on the performance of our cellular genetic algorithm when it was applied to the minimization problem of Schwefel function in Fig. 5 (b). For example, the average function value was 9.479 by the fitness-based algorithm while it was 1.186 by the rank-based algorithm in the case of six elite individuals.

4 Conclusions

In this paper, we examined the effect of local ranking on the performance of our cellular genetic algorithm with the two neighborhood structures. The local ranking of an individual among its competition neighborhood was used as its fitness in the parent selection phase. The effect of using the local ranking was unclear in computational experiments on a knapsack problem. That is, similar results were obtained by the fitness-based and rank-based algorithms. On the contrary, experimental results on Schwefel function clearly showed that the use of the local ranking improved the ability of our cellular genetic algorithm to escape from local optima. This ability is useful especially in the handling of optimization problems with many local optima. No clear negative effect of using the local ranking was observed in our computational experiments. As shown in Fig. 5, the use of the local ranking made our cellular genetic algorithm more robust with respect to the specification of the strength of elitism.

Acknowledgement

This work was partially supported by Grant-in-Aid for Scientific Research (B): KAKENHI (20300084).

References

1. Gorges-Schleuter, M.: ASPARAGOS: An Asynchronous Parallel Genetic Optimization Strategy. In: Proc. of 3rd International Conference on Genetic Algorithms, pp. 422–427 (1989)
2. Manderick, B., Spiessens, P.: Fine-Grained Parallel Genetic Algorithms. In: Proc. of 3rd International Conference on Genetic Algorithms, pp. 428–433 (1989)
3. Spiessens, P., Manderick, B.: A Massively Parallel Genetic Algorithm: Implementation and First Analysis. In: Proc. of 4th International Conference on Genetic Algorithms, pp. 279–286 (1991)
4. Whitley, D.: Cellular Genetic Algorithms. In: Proc. of 5th International Conference on Genetic Algorithms, p. 658 (1993)
5. Alba, E., Dorronsoro, B.: The Exploration/Exploitation Tradeoff in Dynamic Cellular Genetic Algorithms. *IEEE Trans. on Evolutionary Computation* 9, 126–142 (2005)
6. Alba, E., Tomassini, M.: Parallelism and Evolutionary Algorithms. *IEEE Trans. on Evolutionary Computation* 6, 443–462 (2002)
7. Cantu-Paz, E.: *Efficient and Accurate Parallel Genetic Algorithms*. Springer, Berlin (2000)

8. Giacobini, M., Tomassini, M., Tettamanzi, A.G.B., Alba, E.: Selection Intensity in Cellular Evolutionary Algorithms for Regular Lattices. *IEEE Trans. on Evolutionary Computation* 9, 489–505 (2005)
9. Slatkin, M., Wilson, D.S.: Coevolution in Structured Demes. *Proc. of National Academy of Sciences* 76, 2084–2087 (1979)
10. Wilson, D.S.: Structured Demes and the Evolution of Group-Advantageous Traits. *The American Naturalist* 111, 157–185 (1977)
11. Wilson, D.S.: Structured Demes and Trait-Group Variation. *The American Naturalist* 113, 606–610 (1979)
12. Ifti, M., Killingback, T., Doebeli, M.: Effects of Neighborhood Size and Connectivity on the Spatial Continuous Prisoner's Dilemma. *Journal of Theoretical Biology* 231, 97–106 (2004)
13. Ishibuchi, H., Namikawa, N.: Evolution of Iterated Prisoner's Dilemma Game Strategies in Structured Demes under Random Pairing in Game Playing. *IEEE Trans. on Evolutionary Computation* 9, 552–561 (2005)
14. Ishibuchi, H., Doi, T., Nojima, Y.: Effects of Using Two Neighborhood Structures in Cellular Genetic Algorithms for Function Optimization. In: Runarsson, T.P., Beyer, H.-G., Burke, E.K., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) *PPSN 2006. LNCS*, vol. 4193, pp. 949–958. Springer, Heidelberg (2006)
15. Ishibuchi, H., Tsukamoto, N., Nojima, Y.: Examining the Effect of Elitism in Cellular Genetic Algorithms using Two Neighborhood Structures. In: Rudolph, G., Jansen, T., Lucas, S., Poloni, C., Beume, N. (eds.) *PPSN 2008. LNCS*, vol. 5199. Springer, Heidelberg (2008)
16. Zitzler, E., Thiele, L.: Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Trans. on Evolutionary Computation* 3, 257–271 (1999)

Information Theoretic Classification of Problems for Metaheuristics

Kent C.B. Steer, Andrew Wirth, and Saman K. Halgamuge

The University of Melbourne, Parkville, Australia
`k.steer1@pgrad.unimelb.edu.au`

Abstract. This paper proposes a model for metaheuristic research which recognises the need to match algorithms to problems. An empirical approach to producing a mapping from problems to algorithms is presented. This mapping, if successful, will encapsulate the knowledge gained from the application of metaheuristics to the spectrum of real problems. Information theoretic measures are suggested as a means of associating a dominant algorithm with a set of problems.

Keywords: Metaheuristics, information theory, optimisation, problem classification.

1 Introduction

A *metaheuristic* can be described as a general purpose algorithm for solving a range of problems using little or no problem specific knowledge [1]. Examples include evolutionary algorithms (EAs), particle swarm optimisation (PSO), simulated annealing (SA) and tabu search. These techniques have been successful in solving real world problems from a variety of fields [2]. When faced with a new problem the practitioner must decide which metaheuristic to use and how to configure it. The degree of difficulty of these decisions will depend largely on the performance requirements accompanying the given problem. Nonetheless, this process could benefit from a better understanding of the relationship between problems, algorithms and performance [3]. Ideally, for a given problem and performance measure, the practitioner would use some low cost process to determine the best algorithm.

We define *real problems* as those problems whose solution is of interest beyond the research community. The *encoding* of a real problem will determine the problem structure, which in turn will influence algorithm performance. However, for this paper we will assume black-box functions so that the task of selecting a good encoding can be ignored.

While it has been argued that the No Free Lunch Theorems (NFLTs) are not applicable to the set of real problems [4], [5], they do emphasise the need to match algorithms to problems. That is, to achieve better than random performance we must utilise problem-specific knowledge in algorithm selection [6]. In a broad sense, this may occur implicitly when the success of metaheuristics on

real problems acts as feedback to the research community. However, we are not just interested in metaheuristics that perform better than random search over the set of real problems. For a given problem, we would like to know the best known metaheuristic. Furthermore, the number of metaheuristics currently in use supports the claim that there is no universal problem solver, even when we restrict our attention to real problems.

In this paper a model for algorithm research which may assist algorithm selection for real problems will be described. It is hoped that this approach will reduce the human time investment required when attempting to solve a new problem. Additionally, for problems beyond the scope of expert analysis—or when no expert is thought to exist [7]—this model is an alternative to simple trial-and-error. This model attempts to classify problems using measures designed to differentiate algorithm performance.

A number of measures based on information characteristics proposed by Vassilev *et al.* [3] will be explored as a possible classification scheme. These measures were designed to capture features of problem structure thought to be related to algorithm performance.

The paper is organised as follows. In Section 2 we introduce the notation that will be used throughout this paper. Section 3 describes the concept of a *dominant algorithm set*, which provides the backbone of the classification model proposed in Section 4. The information theoretic problem measures are presented in Section 5 and demonstrated for a number of popular benchmark functions. Concluding remarks and suggestions for future work are offered in Section 7.

2 Definitions

Following [6] and [8], we now provide some definitions. We will restrict ourselves to combinatorial optimisation, and assume an optimisation problem f is a mapping from the search space $\mathcal{X}$ into the cost space $\mathcal{Y}$. Formally, $f : \mathcal{X} \mapsto \mathcal{Y}$, where f is taken from the space of all possible combinatorial optimisation problems $\mathcal{F}$. The spaces $\mathcal{X}$ and $\mathcal{Y}$ are assumed to be finite, and $|\mathcal{F}| = |\mathcal{Y}|^{|\mathcal{X}|}$.

An algorithm $a \in \mathcal{A}$ is a mapping from a time-ordered sample of the problem space d_m to a single new point $x \in \mathcal{X}$, where $\mathcal{A}$ is the space of all possible algorithms. The sample contains all distinct points explored by the algorithm, $d_m \equiv \{(d_m^x(1), d_m^y(1)), \dots, (d_m^x(m), d_m^y(m))\}$, where $d_m^x(i) \in \mathcal{X}$ is the i th point explored and $d_m^y(i) = f(d_m^x(i))$ is the corresponding cost value.

3 Dominant Algorithm Sets

One implication of the NFLTs [6] is that we can only talk about the performance of algorithms with respect to some specific problem or set of problems. It is often implied that we are talking about the set of problems which arise from real world scenarios. However, general statements about the performance of some algorithm over this poorly defined set are not likely to be well justified. To

compare two algorithms we need to be clear about the set of problems over which the comparison is being made.

Consider the set of all known algorithms $\mathcal{A}_k \subset \mathcal{A}$, which we will assume to be static. Each algorithm $a \in \mathcal{A}_k$ can be associated with a subset $F_a \subset \mathcal{F}$, over which a is the dominant algorithm. An algorithm is said to be dominant over a problem set if it performs no worse than any other for all problems in that set, for a given performance measure. That is, a_d is dominant over F_a if and only if

$$\varphi(a_d, f) \geq \varphi(a, f) \quad \forall f \in F_a, a \in \mathcal{A}_k, \quad (1)$$

for some performance measure $\varphi(a, f)$. Accordingly, we can define the classification function

$$\mathcal{Z}(f) = \begin{cases} a_1, & \text{if } f \in F_{a_1} \\ a_2, & \text{if } f \in F_{a_2} \\ \vdots & \end{cases} \quad (2)$$

which maps problems to algorithms. It is possible that two distinct algorithms will perform equally well for certain problems, therefore the problem subsets F_{a_i} are not mutually exclusive, but they do cover $\mathcal{F}$.

For a given problem f , if we know an F_a such that $f \in F_a$, then $\mathcal{Z}$ would immediately indicate a most suitable algorithm. This premise will be explored in the next section.

4 Dominant Algorithm Problem Classification Model for Algorithm Research

Algorithm research can be seen as two distinct yet complementary endeavours. The first places emphasis on discovering new algorithms, and improving the performance of existing ones over a range of problems; we can call this *algorithm science*. The second places emphasis on solving specific problems; we can call this *algorithm engineering*. Algorithm engineering will typically draw on knowledge provided by algorithm science, but will also innovate and discover new ways of exploiting specific problem characteristics. From this perspective we propose a model for algorithm research in which the aim of algorithm science is to produce $\mathcal{Z}$, while the aim of algorithm engineering is to use $\mathcal{Z}$ to solve real problems.

4.1 Relaxing the Definition of $\mathcal{Z}$

The definition of $\mathcal{Z}$ given in (2) requires a separate dominant problem set for every algorithm. A more useful definition would aim to group algorithms to form larger subsets. For example, we could define the set $\mathcal{F}_{SA}$ such that for all $f \in \mathcal{F}_{SA}$ there is some $a \in A_{SA}$ that is the dominant algorithm, where A_{SA} is the set of all simulated annealing variants. This leads to the relaxed classification function

$$\mathcal{Z}'(f) = \begin{cases} A_1, & \text{if } f \in F_{A_1} \\ A_2, & \text{if } f \in F_{A_2} \\ \vdots & \end{cases} \quad (3)$$

Let $\mathcal{A}_c = \{A_1, A_2, \dots, A_l\}$ be the set of all chosen algorithm sets. The task of producing $\mathcal{Z}'$ exactly for a given $\mathcal{A}_c$ and $\mathcal{F}$ is daunting. We now discuss an empirical approach to obtaining a function $\psi : \mathcal{F} \mapsto \mathcal{A}_c$ which approximates $\mathcal{Z}'$.

4.2 Empirically Derived Classification Set

To begin, we would like to reduce the problem space $\mathcal{F}$ to some lower dimensional measurement space $\mathcal{M}$, whose dimensions are measures which can be computed by sampling a given problem. These measures should be selected to capture features of problems which lead to differences in algorithm performance. One possible set of measures is presented in Section 5.

Once this set of measures $\phi : \mathcal{F} \mapsto \mathcal{M}$ has been selected, a sample set of problems $\mathcal{F}_s \subset \mathcal{F}$ should be chosen such that they map to distinct regions in $\mathcal{M}$. These need not be evenly distributed throughout $\mathcal{M}$, but rather, to be of practical use, they should be concentrated in regions that correspond to real problems. However, a well designed measurement may produce $\mathcal{M}$ for which the space resolution is biased towards real problems. In this case a uniform distribution would be desirable.

For a given performance measure φ , each $A \in \mathcal{A}_c$ should be evaluated with respect to each $f \in \mathcal{F}_s$, where evaluation consists of finding the greatest value of φ for all $a \in A$. These evaluations can be used to determine a *winner* for each problem. The winner becomes the label for a training example, where the inputs are the measurements for the given problem. In this way a training set can be constructed with each example derived from a unique problem $f \in \mathcal{F}_s$. Some supervised learning procedure can then be used to obtain ψ .

When faced with a new problem $f \in \mathcal{F}$, the measurements are computed and used as inputs to ψ . Formally, $\psi : \mathcal{M} \mapsto \mathcal{A}_c$. This provides the algorithm engineer with the set A of algorithms likely to give the best performance for the new problem. The difficulty of finding the best algorithm $a \in A$ depends on how the members of $\mathcal{A}_c$ are defined. Continuing from the simulated annealing example, if $\psi(\phi(f)) = A_{SA}$, then this task would amount to cooling rate adjustments.

If ψ is represented as a decision tree, then only the measures along one path need to be computed. Additionally, the structure of the tree would provide some insight into the utility of chosen problem measures.

5 Information Theoretic Problem Measures

Initially proposed by Wright [9], the *fitness landscape* is a way of visualising a search space. A problem f can be seen as a landscape where variables x define the position, the objective function $y = f(x)$ defines the height—or fitness—for each position, and constraints define the boundaries. Optimisation involves finding the highest (or lowest) point in this landscape. This analogy helps us to understand the differences in various heuristic search procedures, as well as the measurements.

Vassilev *et al.* [3] have developed measures which they claim reflect various distinct landscape characteristics: *ruggedness*, *modality* and *stability*. These measures and their computation are summarised here.

5.1 Information Content

To compute the information content of a landscape f a sequence of N fitness values d_N^y is obtained by performing a random walk a_W . Thus, if $\mathcal{X} \subset \mathbb{R}^n$, then the i th point of the walk $x_i \in \mathcal{X}$ is generated by

$$x_i \equiv \begin{bmatrix} x_i(1) \\ x_i(2) \\ \vdots \\ x_i(n) \end{bmatrix} = \begin{bmatrix} x_{i-1}(1) + r_i(1)\xi \\ x_{i-1}(2) + r_i(2)\xi \\ \vdots \\ x_{i-1}(n) + r_i(n)\xi \end{bmatrix} \quad (4)$$

where $r \sim U(-1, 1)$ is a uniformly distributed random variable, and ξ is the maximum step size. The resultant sequence of fitness values is then

$$d_N^y = \{f(x_1), f(x_2), \dots, f(x_N)\}. \quad (5)$$

The differences between consecutive values in this fitness sequence are then used to produce a string $S(\varepsilon) = s_1 s_2 s_3 \dots s_N$ of symbols $s_i \in \{\bar{1}, 0, 1\}$ given by

$$s_i(\varepsilon) = \Psi(i, \varepsilon) \quad (6)$$

where

$$\Psi(i, \varepsilon) = \begin{cases} \bar{1}, & \text{if } d_N^y(i) - d_N^y(i-1) < -\varepsilon \\ 0, & \text{if } |d_N^y(i) - d_N^y(i-1)| \leq \varepsilon \\ 1, & \text{if } d_N^y(i) - d_N^y(i-1) > \varepsilon \end{cases} \quad (7)$$

for any fixed $\varepsilon > 0$. The sensitivity parameter ε is used to control the degree of detail obtained. It can also be thought of as a scaling factor. The information content, which characterises the ruggedness of the landscape, is then given by

$$H(\varepsilon) = - \sum_{p \neq q} P_{[pq]} \log_6 P_{[pq]}, \quad (8)$$

where $p, q \in \{1, 0, \bar{1}\}$, and $P_{[pq]}$ is the frequency of the substring pq in the symbol string $S(\varepsilon)$. The number of substrings pq where $p \neq q$ is 3!, hence, the logarithm is taken with base six so that the information content is in the interval $[0, 1]$. We recall that as for entropy we let $0 \log_6 0 \equiv 0$.

5.2 Partial Information Content

The partial information content is a measure devised by Vassilev *et al.* [3] to characterise the modality of the landscape. A new string $S'(\varepsilon)$ is constructed from $S(\varepsilon)$ by removing 0's and repeated symbols

$$S'(\varepsilon) = s_{i_1} s_{i_1} \dots s_{i_\mu}, \quad (9)$$

where $s_{i_j} \neq 0, s_{i_j} \neq s_{i_{j-1}}$ and $j > 1$. This modified string has the form "...1 $\bar{1}$ 1 $\bar{1}$ 1...". The partial information content $M(\varepsilon)$ is then defined as

$$M(\varepsilon) = \frac{\mu}{N}, \quad (10)$$

where μ is the length of the modified string and N is the length of the original string, such that $0 \leq M(\varepsilon) \leq 1$.

Since μ is a measure of the number of times the fitness sequence changed direction by more than ε , the number of optima encountered N_O is equal to the number of '1 $\bar{1}$ ' pairs,

$$N_O = \left\lfloor \frac{\mu}{2} \right\rfloor = \left\lfloor \frac{NM(\varepsilon)}{2} \right\rfloor. \quad (11)$$

5.3 Information Stability

The information stability ε^* of a landscape is defined to be the smallest value of ε such that $|d_N^y(i) - d_N^y(i-1)| \leq \varepsilon, \forall i > 1$. In practice this translates to the maximum absolute change in fitness encountered during the walk,

$$\varepsilon^* = \max_{i=2}^N |d_N^y(i) - d_N^y(i-1)|. \quad (12)$$

This measure gives an indication of the scale of problem; the largest change in fitness likely to be encountered.

5.4 Secondary Measures

When the aforementioned information measures are computed for some problem, it is necessary to choose a value of the sensitivity parameter ε . However, this can be a difficult decision to make in advance. Accordingly, we introduce the following secondary measures: *maximum information content* $H_{\max}$, *settling sensitivity* ε_s , *initial partial information content* M_0 , and *half partial information content point* $\varepsilon_{0.5}$, given by

$$H_{\max} = \max_{\varepsilon} \{H(\varepsilon)\}, \quad (13)$$

$$\varepsilon_s = \min_{\varepsilon} \{\varepsilon : H(\varepsilon) < 0.05\}, \quad (14)$$

$$M_0 = M(\varepsilon = 0), \text{ and} \quad (15)$$

$$\varepsilon_{0.5} = \max_{\varepsilon} \{\varepsilon : M(\varepsilon) > 0.5M_0\}. \quad (16)$$

These can be found by graphing $H(\varepsilon)$ and $M(\varepsilon)$ against ε as can be seen in Figs. 1-3 for three benchmark functions (Table 1), where $n = 10$. The data shown is averaged over 500 random walks (4) of length 1000 for each maximum step size $\xi = \{0.05, 0.2, 0.45, 0.8\}$. This amount of computation would not generally be required, but has been done here to clearly demonstrate the shape of the curves.

Table 1. Benchmark functions used to demonstrate information measures

Function	$f(x)$
<i>Sphere</i>	$f(x) = \sum_{j=1}^n x(j)^2$
<i>Ackley</i>	$f(x) = 20 - 20 \exp \left\{ -0.2 \left\{ \frac{1}{n} \sum_{j=1}^n x(j)^2 \right\}^{\frac{1}{2}} \right\} \\ + e - \exp \left\{ \frac{1}{n} \sum_{j=1}^n \cos(2\pi x(j)) \right\}$
<i>Rastrigin</i>	$f(x) = \sum_{j=1}^n \{x(j)^2 - 10 \cos(2\pi x(j)) + 10\}$

The secondary problem measures (13-16) are presented in Table 2. The authors wish to emphasise that these results are not intended, nor are they sufficient, to demonstrate the success of this approach. However, by virtue of their variation across the benchmark problems, these measures exhibit the potential to distinguish between problems.

Table 2. Secondary problem measures for selected benchmark problems ($\xi = 0.05$)

Function	$H_{\max}$	ε_s	M_0	$\varepsilon_{0.5}$
<i>Sphere</i>	0.8126	2.88	0.50	0.72
<i>Ackley</i>	0.8117	0.16	0.50	0.03
<i>Rastrigin</i>	0.8736	82.42	0.68	25.15

6 Information Theoretic Classification Example

We will now sketch a hypothetical sequence of decisions and actions leading to the selection of an algorithm family for a specific problem.

Suppose we choose $\phi = \{H_{\max}, \varepsilon_s, M_0, \varepsilon_{0.5}\}$ as our set of problem measures, and some sample problems $\mathcal{F}_s = \{f_1, f_2, \dots, f_n\}$ to begin charting the measurement space. We then define the algorithm families which together form the chosen algorithm set $\mathcal{A}_c$. For each family the base algorithm and the allowable variations should be specified unambiguously. For this example we will choose just two families, simulated annealing A_{SA} and particle swarm optimisation A_{PSO} .

A performance measure φ must also be selected to compare algorithms in a useful way. Here we will assume computation time is of importance, and that the function evaluations are relatively expensive, so we define

$$\varphi(a, f) = \min_i \{d_m^y(i) : i = 1 \dots m\}, \quad (17)$$

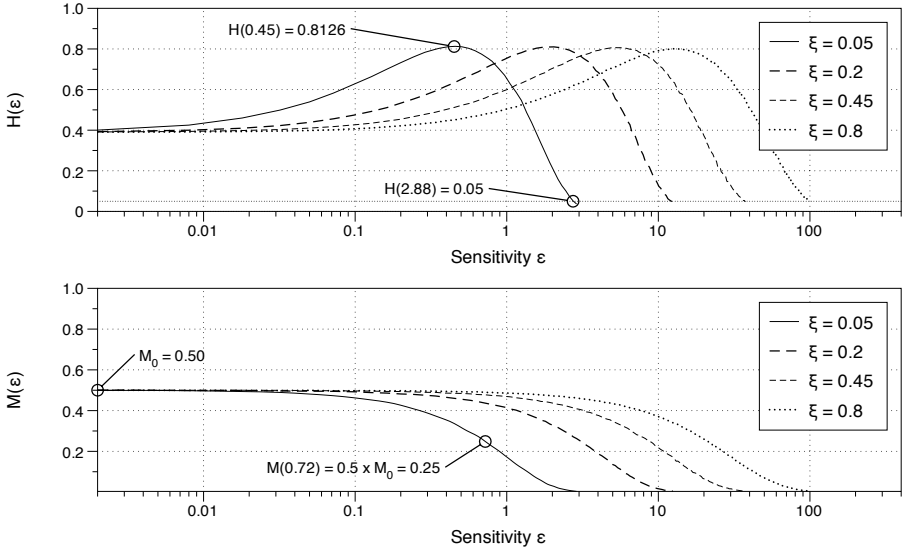


Fig. 1. Information Content of Sphere function

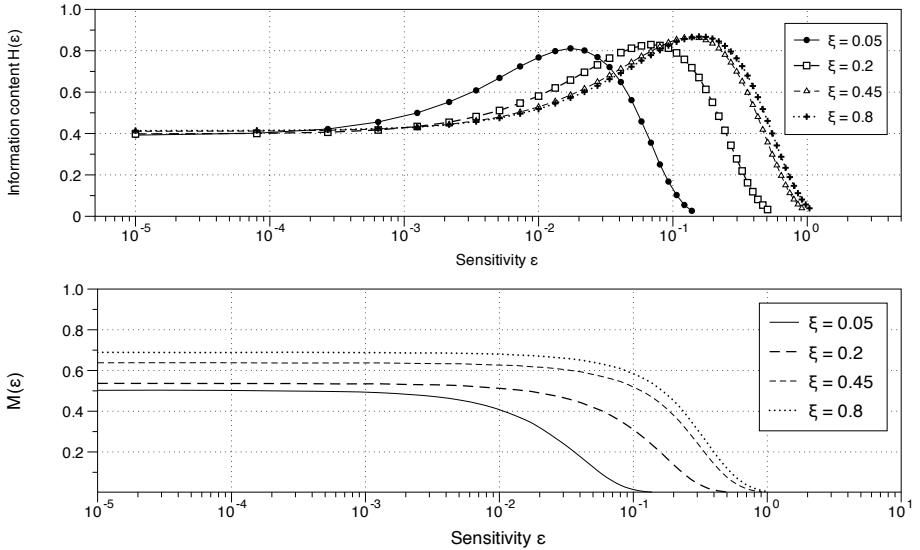


Fig. 2. Information Content of Ackley's function

where m is fixed, and d_m^y is the sequence of function evaluations produced by a on f . For non-deterministic search procedures it is necessary to average this performance over enough runs to make the results statistically significant [2].

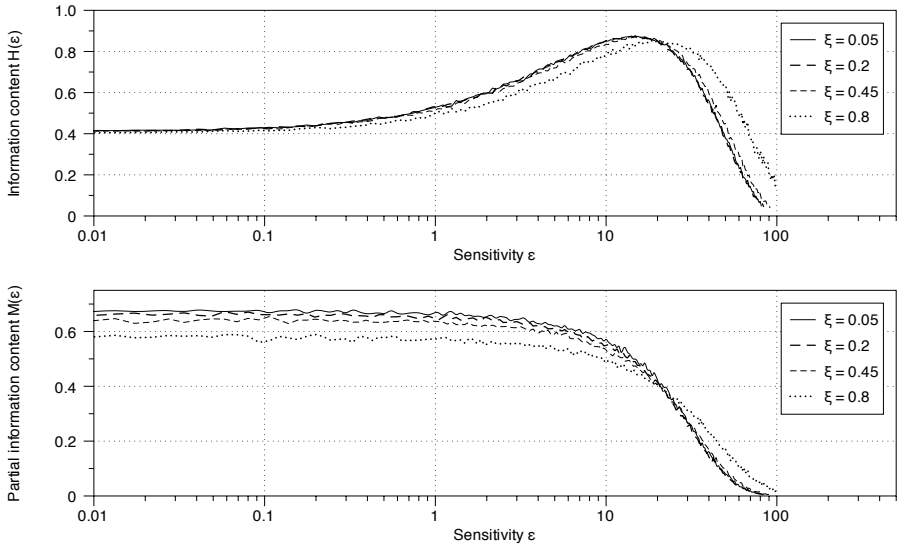


Fig. 3. Information Content of Rastrigin function

We then apply each algorithm family to each problem, allowing for tuning within the family to find the performance of the best member. The dominant algorithm family for each problem is then determined by comparing the respective performance measures.

The training set can then be produced, where each row in Table 2 is an input example, and the corresponding dominant algorithm families are the labels. A neural network classifier, for example, could then be trained and provided to an algorithm engineer trying to solve some new problem. The new problem would be interrogated to produce the previously specified measures ϕ , which would be fed into the classifier, thereby computing the best algorithm family for that problem.

7 Conclusions and Future Work

A model for the classification of optimisation problems has been proposed. This model, if successful, would provide an evidence based method for algorithm selection for new problems. A possible empirical approach to problem classification has been presented using information theoretic measures, which are thought to be related to algorithm performance.

The ideas presented here are largely speculative, and as such, more research is required before any solid conclusions can be drawn. In particular, the orderly structure of the problem space—with respect to dominant algorithms—is a necessary prerequisite for an advantage to be gained by this approach. Accordingly,

the authors are currently investigating the correlations between problem measures presented here and the performance of various metaheuristics.

Rather than leaving the task of selecting the best $a \in A$ to the algorithm engineer, a similar approach to the one presented here for algorithm family selection can be used for parameter tuning. Additionally, the extension of this approach to dynamic problems may offer an alternative to current adaptive algorithms.

References

1. Borenstein, Y., Poli, R.: Information Perspective of Optimization. In: Runarsson, T.P., Beyer, H.-G., Burke, E.K., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) PPSN 2006. LNCS, vol. 4193, pp. 102–111. Springer, Heidelberg (2006)
2. Dréo, J., Pérowski, A., Siarry, P., Taillard, E.: Metaheuristics for Hard Optimization: Methods and Case Studies. Springer, Heidelberg (2006)
3. Vassilev, V.K., Fogarty, T.C., Miller, J.F.: Information characteristics and the structure of landscapes. *Evolutionary Computation* 8, 31–60 (2000)
4. Koehler, G.J.: Conditions that obviate the no-free-lunch theorems for optimization. *Inform. Journal on Computing* 19, 273–279 (2007)
5. Igel, C., Toussaint, M.: On classes of functions for which no free lunch results hold. *Inf. Process. Lett.* 86, 317–321 (2003)
6. Wolpert, D.H., MacReady, W.G.: No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1, 67–82 (1997)
7. Aytug, H., Bhattacharyya, S., Koehler, G., Snowdon, J.: A review of machine learning in scheduling. *IEEE Transactions on Engineering Management* 41, 165–171 (1994)
8. Ho, Y.C., Pepyne, D.L.: Simple explanation of the no-free-lunch theorem and its implications. *Journal of Optimization Theory and Applications* 115, 549–570 (2002)
9. Wright, S.: Evolution in mendelian populations. *Bulletin of Mathematical Biology* 52, 241–295 (1990)

Task Decomposition for Optimization Problem Solving

Ehab Z. Elfeky, Ruhul A. Sarker, and Daryl L. Essam

School of ITEE, University of New South Wales at ADFA, Canberra, Australia
{e.elfeky, r.sarker, d.essam}@adfa.edu.au

Abstract. This paper examines a new way of dividing computational tasks into smaller interacting components, in order to effectively solve constrained optimization problems. In dividing the tasks, we propose problem decomposition, and the use of GAs as the solution approach. In this paper, we consider problems with block angular structures with or without overlapping variables. We decompose not only the problem but also appropriately the chromosome for different components of the problem. We also design a communication process for exchanging information between the components. The approach can be implemented for solving large scale optimization problems using parallel machines. A number of test problems have been solved to demonstrate the use of the proposed approach. The results are very encouraging.

1 Introduction

The definition of ‘large-scale’ varies among researchers. Some researchers define it in terms of the number of variables and/or constraints of a problem, while others consider it as the complexity of problems (even for problem with small numbers of variables and constraints). The first type can also be called computer dependent, as it depends mainly on the capabilities of the computer, on the other hand, the second type is recognized as problem dependent [1], which could be because of the nature of the constraints and/or objective functions (such as non-linear, multimodal, etc.) or the structure of the problem (such as decomposable or not). Researchers may also classify large-scale optimization problems depending upon the type of the variables involved, such as continuous or integer or nonlinear. As the integer and nonlinear optimization models are difficult to solve, such models with a couple of hundreds of variables may be considered as large scale. On the other hand, a linear programming model (of course limited to continuous variables) with 20 or 30 times more variables may not be considered as a large scale problem.

In this study, we consider constrained optimization problems with continuous variables. In addition, we assume the problems contain large numbers of variables and/or constraints, and that the finding of the exact optimal solution for some small-scale nonlinear functions (considered as objective function) is not an easy task [2], it is much more difficult to find the optimal solution for nonlinear functions with many variables. Therefore, the main objective in large-scale optimization is to find an acceptable solution within a reasonable time limit. The addition of functional constraints to large problems makes the problem even more challenging.

The problem we consider in this paper can be stated as follows:

$$\begin{aligned}
 P1: \min \quad & f(X) \\
 \text{subject to} \quad & g_i(X) \leq 0, \quad i = 1, 2, \dots, m, \quad h_j(X) = 0, \quad j = 1, 2, \dots, p; \\
 & L_i \leq x_i \leq U_i, \quad i = 1, 2, \dots, n.
 \end{aligned}$$

where $X \in R^n$ is the vector of solutions $X=[x_1, x_2, \dots, x_n]$. The objective function is $f(x)$, m is the number of inequality constraints, $g_i(x)$ is the i th inequality constraint, p is the number of equality constraints, and $h_j(x)$ is the j th equality constraint. Each decision variable x_i has a lower bound L_i and an upper bound U_i .

Over the last few decades, researchers and practitioners have introduced different approaches for solving large scale problems. In the traditional optimization domain, these methods include decomposition approaches and problem specific heuristics. In the classical decomposition approaches, the problem is divided into a number of smaller sub-problems by exploiting the problem structure and then solving each of them independently [3].

The acceptability of applying evolutionary computation (EC) as a tool for solving large optimization problems has increased as EC methods do not require that certain mathematical properties of functions, such as convexity, differentiability and continuity, need to be satisfied. Also, EC does not require any pre-knowledge about the search space. Most of the researchers effort in solving the decomposable large-scale optimization problems using EC is applicable only to certain classes of mathematical programming models such as multidimensional 0-1 knapsack by Kato and Sakawa [4], mesh-interconnected systems by [5], mixed-integer nonlinear programming by [6], and unconstrained optimization by yang et.al. [7]. The decomposition way in these papers seems to be the same; but actually there are minor differences due to the specialty in the problem structure. Another feature could be attractive in using EC, that it is easy to be designed for parallel processing [3]. Parallel processing reduces the length of computational duration (not necessarily the overall computational time) as the computational tasks are divided and processed concurrently. However, the approach used for allocating the computational tasks to the multiple processors and their appropriate integration is vital for effective parallel processing.

This paper introduces a new way of dividing computational tasks for a class of constrained optimization problems that will facilitate parallel processing if required. We use a genetic algorithm (GA) for solving a class of problems known as block angular structure problems. Based on the problem structure, we decompose the problem into a number of sub-problems and one common problem. We also decompose the chromosomes as only a portion is required by each sub-problem. The sub-problems are maintaining the feasibility, while the common problem is improving the optimality and maintaining the overall problem feasibility. This way of dividing the volume of computational tasks, and distributing the tasks for maintaining feasibility and improving fitness, should reduce the complexity of the problem solving process. However, it requires to be designed with a communication process for the exchange of information between the common problem and the sub-problems. The details of this approach are presented later on in the paper. We have solved a number of test problems to demonstrate the use of the proposed methodology. The results are very encouraging for later experimentation in a distributed environment.

This paper is organized as follows: section 2 presents the problem structure and the decomposition method, while the main algorithm is proposed in section 3. A numerical example is illustrated in section 4. Section 5 explains some experiments and its results. The conclusions are stated in section 6. And finally the references are stated at the end of the paper.

2 The Problem Structure and Decomposition

As discussed in section II, certain classes of problems can then be decomposed into a number of smaller sub-problems, which can provide computational advantages, when solving with classical optimization techniques. However, these decomposition techniques cannot be applied to all classes of optimization problems. In contrast, GAs can be used to potentially solve any or all classes of problems. As in section III, by appropriately partitioning the problem and by then processing with parallel processors, PGAs are useful to reduce the computational time and computer memory required.

In this research, our objective is to partition the problem into a number of smaller problems and solve the problems by solving the sub-problems in a coordinated fashion. This concept can be implemented easily for parallel processing which is in a way similar to PGAs. The problem under consideration could belong to any class of optimization problems.

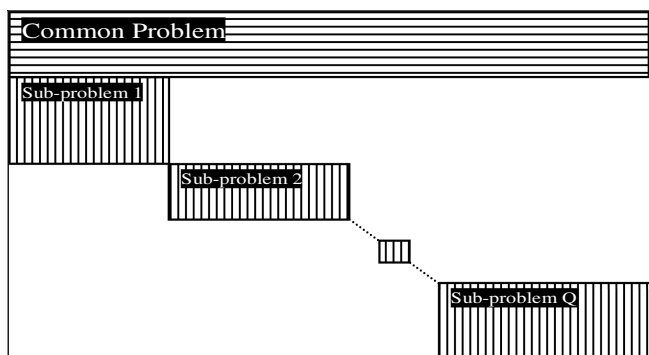


Fig. 1. Block Angular Problem Structure without overlapping variables

In many large scale practical optimization problems, certain (or a certain sub-set of) constraints involve a certain sub-set of variables. In these cases, the constraints can be grouped based on the variables involved. A typical example is a multidivisional problem. We consider a specially structured (block angular) problem, where the variables can be rearranged to be part of a smaller common problem and Q sub-problems as shown in Figure 1. In this figure, the columns represent the variables, and the rows represent the constraints and the objective function. The common problem in Fig. 1 contains the objective function and a subset of constraints (SC_c). The common problem would contain all or most of the variables. We assume the common problem will have constraints that are non separable in terms of variables. Each sub-problem consists of a subset of constraints (SC_k) which involve a subset of variables (SV_k).

Here, $V = SV_1 \cup SV_2 \cup \dots \cup SV_Q = \text{complete set of variables}$

$C = SC_c \cup SC_1 \cup \dots \cup SC_Q = \text{complete set of constraints}$

The intersection of any two subsets of variables and any two subsets of constraints is an empty set (excludes SC_c). That means,

$$SV_i \cap SV_j = \emptyset \quad \text{for } i \neq j \quad \text{and} \quad SC_i \cap SC_j = \emptyset \quad \text{for } i \neq j$$

Each sub-problem consists of decision variables (n_{sub_q}) and constraints (equality (p_{sub_q}) and inequality (m_{sub_q})). The common problem contains all the decision variables (n) or most variables, and the remaining constraints (p_c equality and m_c inequality) that cannot be broken down according to the current decomposition. The objective function of the problem is neither changed nor decomposed, and exists in the common problem in its original form.

3 The Proposed Algorithm

As discussed earlier, the common problem will deal with the whole chromosome and its task is to optimize the fitness function while maintaining its internal feasibility. Each sub-problem will deal with its corresponding sub-chromosomes and its task is to find feasible sub-chromosomes. Since the common problem and the sub-problems are solved separately, there must be some forms of information exchange between these problems so that the solutions can gradually be improved while maintaining feasibility. We assume that the common problem and the sub-problems will exchange information after evolving every few generations independently, through a star shape communication topology. The common problem will receive and send sub-chromosomes from and to all the sub-problems, but no sub-problem is allowed to interact with another sub-problem explicitly.

The communication between the common problem and the sub-problems occurs in two ways: Forward Migration and Back Migration. Forward migration is the transference of the chromosomes from the common problem to the sub-problems, while breaking them down into sub-chromosomes. Backward migration occurs in the opposite direction, from the sub-problems to the common problem, while combining the sub-chromosomes up to a single chromosome. This topology is depicted in Figure 2.

The backward migration is choosing only the feasible sub-chromosomes randomly; hence, the challenging task here is of how to keep the feasibility of the migrant individuals in the common problem. Preserving the sub-chromosomes building blocks is the solution; thus the crossover in the common problem would only shuffle the sub-chromosomes between parents. Also, there is no mutation to be applied in the common population. The backward migration is the only source of the change in the common population. But this is not enough to compensate the lack of the diversity. Hence, for further diversification, one can inject a certain number of randomly generated sub-chromosomes in the sub-problems.

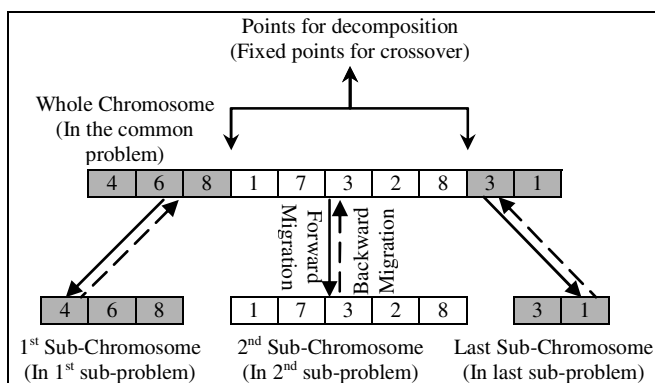


Fig. 2. The communication topology (Inner View)

The sub-problems have three types of sub-chromosomes after forward migration: the first is the original sub-chromosomes; their mission is to preserve the previous direction of the search process. The second type is the sub-chromosomes coming from the common problem, which are guiding the search process towards the global optimality, and the third type is the regenerated individuals which give new blood in the common problem population and which save the algorithm from falling into a local optima.

The steps of the proposed algorithm can be summarized as follows:

- Step 1:** Decompose the problem into one common problem and Q sub-problems
- Step 2:** Start with the sub-problems and evolve the sub-chromosomes population until they reach certain feasibility ratios
- Step 3:** Make up the chromosomes for the common population taking sub-chromosomes from the sub-problems (known as backward migration). Evolve the common problem population for a certain number of generations.
- Step 4:** If the stopping criterion is not satisfied, break certain chromosomes into sub-chromosomes and send them back to the sub-problems (backward migration). Evolve the sub-chromosomes' population.
- Step 5:** Go to Step 3.

In Step 1, we decompose the problem $P1$ into one common problem and Q sub-problems as shown below

$$\begin{aligned}
 P - Comm : & \min f(X) \\
 \text{subject to} & \quad g_i(X) \leq 0, & i = 1, 2, \dots, m_c, \\
 & \quad h_j(X) = 0, & j = 1, 2, \dots, p_c, \\
 & \quad L_i \leq x_i \leq U_i, & i = 1, 2, \dots, n.
 \end{aligned}$$

where m_c is the number of inequality constraints in the common problem, and p_c is the number of equality constraints.

$$\begin{aligned}
P - Sub_q : \min \quad & f(X) = g_i(X) + |h_j(X)| \\
\text{subject to} \quad & i = 1, 2, \dots, m_{sub_q}, j = 1, 2, \dots, p_{sub_q}, \\
& L_i \leq x_i \leq U_i, i = 1, 2, \dots, n_{sub_q}, \\
& q = 1, 2, \dots, Q.
\end{aligned}$$

Such as,

$$\begin{aligned}
m &= m_c + \sum_{q=1}^Q m_{sub_q}, p = p_c + \sum_{q=1}^Q p_{sub_q}, \\
\forall i, i \in n_{sub_a}, i \in n_{sub_b} &\Rightarrow a = b
\end{aligned}$$

where n_{sub_q} is the number of decision variables which are considered in the sub-problem number q , m_{sub_q} is the number of inequality constraints, and p_{sub_q} is the number of equality constraints.

As in Step 2, the sub-problems are evolved until the feasibility ratio exceeds α_q .

Where α_q is the minimum feasibility ratio for migration from the sub-problem q to the common problem? From each sub-problem, choose only feasible sub-chromosomes in order to make up the chromosomes for the common problem. In Step 3, evolve the common problem CP for CMF number of generations. In Step 4, evolve every sub-population Sq for SMF_q number of generations. The stopping criterion is the maximum number of evaluation condition.

4 Numerical Example

To test the performance of the proposed algorithm, we have used four test problems. The first three are from the literature: the first problem (g18) is taken from Himmelblau [8], the second and the third test problems were introduced by Dembo [9]. We have designed the fourth problem.

We identify g18 as EES01 in this paper. To demonstrate the implementation of the proposed algorithm, we consider the test problem EES01 as follows.

$$EES01: \text{Min} f(X) = -0.5(x_1x_4 - x_2x_3 + x_3x_9 - x_5x_9 + x_5x_8 - x_6x_7)$$

$$\begin{aligned}
\text{Subject to,} \quad & g_1(X) = 1 - x_3^2 - x_4^2 \geq 0, & g_2(X) &= 1 - x_9^2 \geq 0, \\
& g_3(X) = 1 - x_5^2 - x_6^2 \geq 0, & g_4(X) &= 1 - x_1^2 - (x_2 - x_9)^2 \geq 0, \\
& g_5(X) = 1 - (x_1 - x_5)^2 - (x_2 - x_6)^2 \geq 0, & g_6(X) &= 1 - (x_1 - x_7)^2 - (x_2 - x_8)^2 \geq 0, \\
& g_7(X) = 1 - (x_3 - x_5)^2 - (x_4 - x_6)^2 \geq 0, & g_8(X) &= 1 - (x_3 - x_7)^2 - (x_4 - x_8)^2 \geq 0, \\
& g_9(X) = 1 - x_7^2 - (x_8 - x_9)^2 \geq 0, & g_{10}(X) &= x_1x_4 - x_2x_3 \geq 0, \\
& g_{11}(X) = x_3x_9 \geq 0, & g_{12}(X) &= -x_5x_9 \geq 0, \\
& g_{13}(X) = x_5x_8 - x_6x_7 \geq 0.
\end{aligned}$$

The optimal solution for this test problem has an objective function value of $f(x^*) = -0.866254038$ and the values of the decision variables are: $x^* = (0.8841292, 0.8841292, 0.4672425, 0.03742076, 0.9992996, 0.8841292, 0.4672425, 0.03742076, 0.26E-19)$. To apply the proposed algorithm, we decompose the above test problem manually as follows:

Common Problem: The common problem contains all the decision variables and the constraints $g_4, g_7, g_8, g_9, g_{10}$, and g_{12} . No subset of these constraints can be separated as a sub-problem. So the common problem can be formulated as follows.

$$EES01-Comm: \text{Min } f(X) = -0.5(x_1x_4 - x_2x_3 + x_3x_9 - x_5x_9 + x_5x_8 - x_6x_7)$$

$$\text{Subject to, } g_4(X) = 1 - x_1^2 - (x_2 - x_9)^2 \geq 0, \quad g_7(X) = 1 - (x_3 - x_5)^2 - (x_4 - x_6)^2 \geq 0,$$

$$g_8(X) = 1 - (x_3 - x_7)^2 - (x_4 - x_8)^2 \geq 0, \quad g_9(X) = 1 - x_7^2 - (x_8 - x_9)^2 \geq 0,$$

$$g_{10}(X) = x_1x_4 - x_2x_3 \geq 0, \quad g_{12}(X) = -x_5x_9 \geq 0.$$

Sub-Problems: The constraints, excluding those that are in *EES01-Comm*, can be decomposed into two sub-problems. The first sub-problem *EES01-Sub₁* contains decision variables x_1, x_2, x_5, x_6, x_7 , and x_8 , with their constraints g_3, g_5, g_6 , and g_{13} as shown below. The second sub-problem *EES01-Sub₂* considers decision variables x_3, x_4 , and x_9 , with their constraints g_1, g_2 , and g_{11} .

EES01-Sub₁ :

$$\text{Min } f(X) = g_3(X) + g_5(X) + g_6(X) + g_{13}(X)$$

$$\text{Subject to, } g_3(X) = 1 - x_5^2 - x_6^2,$$

$$g_5(X) = 1 - (x_1 - x_5)^2 - (x_2 - x_6)^2,$$

$$g_6(X) = 1 - (x_1 - x_7)^2 - (x_2 - x_8)^2,$$

$$g_{13}(X) = x_5x_8 - x_6x_7,$$

EES01-Sub₂ :

$$\text{Min } f(X) = g_1(X) + g_2(X) + g_{11}(X)$$

$$\text{Subject to, } g_1(X) = 1 - x_3^2 - x_4^2,$$

$$g_2(X) = 1 - x_9^2, \quad g_{11}(X) = x_3x_9,$$

Dembo [9] has introduced a set of geometric programming problems, one of those test problems was a mathematical programming model of a 3-stage membrane separation process which he indicated as 'problem 6', Dembo has also introduced another test problem (problem 7) which was an extension of problem 6. These two test problems were used as the second and the third test problem in this paper, however, we identify them as EES02 and EES03 in this paper. The problem EES02 has 13 variables and 13 constraints, and can be decomposed into one common problem and two sub-problems. The common problem has 3 constraints, the first sub-problem has 8 variables and 7 constraints, and the second sub-problem has 5 variables and 3 constraints. The problem EES03 has 16 variables and 19 constraints; it has been decomposed into one common problem with 7 constraints and 4 sub-problems. The sub-problems numbered 1, 2, 3 and 4 have 7, 5, 2, 2 variables and 6, 4, 2 constraints respectively.

We have designed the fourth test problem by combining several small problems from the literature, such as problems g1[10], g2[11], g7[12], and g8[11]. The search space of each of these problems is considered as a sub-problem by neglecting their objective functions. The common-problem is designed using the problem g9 [12].

However, the variables in each sub-problem are a subset of the variables in the common problem and no single variable exists in any two sub-problems. This approach can construct a new test problem which has a larger number of variables and constraints; moreover, it introduces different types of complexity to the sub-problems as already introduced from the literature. We introduce this problem as EES04.

5 Experiments and Results

For each test problem mentioned in the pervious section, 30 runs were carried out, each of them with a different seed number. For each of the common populations we are using a population size of 30, tournament selection of size 2 while copying the best 10% of the individuals to the following generation, the proposed crossover that shuffles the sub-individuals between the individuals, with no mutation, and with applying forward migration every 4 generations and by then transferring 30% of the population. For the sub-problems' populations, we are using a population size of 60 to compensate for the lack of diversity in the common population, tournament selection of size 2 without copying any individuals, single point whole arithmetic crossover with probability of 0.8, the mixed uniform and non-uniform mutation [13] with mutation probability of 0.1, and with applying backward migration in every generation and by then transferring 30% of the individuals to the common population. It is important here to mention that the sub-problems' populations will evolve for only one generation and then it will stop and wait for the common population to finish evolving for 4 generations, then the migration process will be executed. The run will stop after 350,000 evaluations for the objective function.

The results of those runs are summarized and compared with the results of another two existing algorithms in table 1: the first algorithm is NSGAI_{II} which was developed by Deb et. al. [14], the second algorithm, is TC which was developed by Elfeky et. al. [15]. Considering Table I, the proposed algorithm obtained the same solutions as

Table 1. Statistical results of the 30 different seed runs with three algorithms: NSGAI_{II} (proposed by Deb et. al. [14]), TC (proposed by Elfeky et. al. [15]), and the third is the proposed algorithm in this paper.(PD-GA)

Problem	Solver	Best	Median	Mean	St. Dev.	worst
EES01	NSGAI _{II}	-0.861379	-0.829642	-0.77166	0.122273	-0.46934
	TC	-0.865996	-0.8590135	-0.85163	0.026563	-0.73855
	PD-GA	-0.863103	-0.819689	-0.77356	0.10935	-0.50164
EES02	NSGAI _{II}	1.0002	1.0002	1.0002	0	1.0002
	TC	1.0002	1.0002	1.0002	0	1.0002
	PD-GA	1.0002	1.0002	1.0002	0	1.0002
EES03	NSGAI _{II}	154.8267	154.8267	154.8267	0	154.8267
	TC	154.8267	154.8267	154.8267	0	154.8267
	PD-GA	154.8267	154.8267	154.8267	0	154.8267
EES04	NSGAI _{II}	948.9665	949.5468	949.6458	0.429398	950.7336
	TC	949.4648	952.0145	951.8837	1.211326	954.473
	PD-GA	949.1623	951.2566	951.4764	1.128195	953.3126

the other two algorithms for two test problems EES02 and EES03, while all of them got better results than the best reported by Dembo [9]. For the test problem EES01, the proposed algorithm obtained better results than NSGAII but not as good as TC. On the other hand, for test problem EES04, the proposed algorithm has produced better results than TC but not as good as NSGAII.

The main purpose of this comparison is to validate the idea behind our algorithm. From the results and the comparison, it can be claimed that the proposed algorithm is working well.

6 Conclusions

In this paper, we have introduced a new approach in order to solve decomposable large-scale constrained optimization problems in the continuous domain. We have considered pre-decomposed problems based on a block angular structure. The main idea was to rely on decomposing both the problem and the chromosomes into sub-problems and sub-chromosomes, and the way of distributing the tasks and the communication process so as to break down the complexity of maintaining feasibility while solving large-scale problems. We have solved four test problems and compared the results with two existing solvers, and the results show that the algorithm is performing well and is competitive with the other solvers.

For future work, as this paper could be considered as an introductory paper, this gives more scope for additional research and ideas to enrich the algorithm. The success of this algorithm encourages us to implement it in a physically distributed environment and to use it for solving further large test problems.

References

- [1] Conn, A.R., Gould, N.I.M., Toint, P.L.: Large-scale nonlinear constrained optimization: a current survey. In: Shanno, D.F., Dixon, L., Spedicato, E. (eds.) *Algorithms for continuous optimization: the state of the art*, vol. 434, pp. 287–332. Kluwer Academic Publishers Group (1994)
- [2] Elfeky, E.Z., Sarker, R.A., Essam, D.L.: Analyzing the Simple Ranking and Selection Process for Constrained Evolutionary Optimization. *Journal of Computer Science And Technology* 23(1), 19–34 (2008)
- [3] Martin, R.K.: *Large Scale Linear and Integer Optimization: A Unified Approach*. Springer, Heidelberg (1998)
- [4] Kato, K., Sakawa, M.: Genetic algorithms with decomposition procedures for multidimensional 0-1 knapsack problems with block angular structures. *IEEE Transactions on Systems, Man, and Cybernetics, Part B* 33(3), 410–419 (2003)
- [5] Lin, S.-S., Chang, H.: A Decomposition-Technique-Based Algorithm for Nonlinear Large Scale Mesh-Interconnected System and Application. *IEICE Trans. Fundamentals* E89-A(10), 2847–2856 (2006)
- [6] Benjamin, W.W., Yixin, C., Andrew, W.: Constrained Global Optimization by Constraint Partitioning and Simulated Annealing. In: *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence*. IEEE Computer Society, Los Alamitos (2006)

- [7] Yang, Z., Tang, K., Yao, X.: Large scale evolutionary optimization using cooperative co-evolution. *Information Sciences* 178(15), 2985–2999 (2008)
- [8] Himmelblau, D.M.: *Applied Nonlinear Programming*. McGraw-Hill, New York (1972)
- [9] Dembo, R.S.: A set of geometric programming test problems and their solutions. *Mathematical Programming* 10(1), 192–213 (1976)
- [10] Floudas, C.A., Pardalos, P.M.: *A Collection of Test Problems for Constrained Global Optimization Algorithms*. LNCS, vol. 455. Springer, Heidelberg (1990)
- [11] Koziel, S., Michalewicz, Z.: Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization. *Evolutionary Computation* 7(1), 19–44 (1999)
- [12] Hock, W., Schittkowski, K.: *Text examples for nonlinear programming codes*. Springer, New York (1981)
- [13] Elfekey Ehab, Z., Sarker Ruhul, A., Essam, D.L.: Analyzing the Simple Ranking and Selection Process for Constrained Evolutionary Optimization. *Journal of Computer Science And Technology* 23(1), 19–34 (2008)
- [14] Deb, K., Agrawal, S., Pratab, A., Meyarivan, T.: A Fast and Elitist Multi-Objective Genetic Algorithm: NSGA-II. *IEEE Trans. on Evolutionary Computation* 6(2), 182–197 (2002)
- [15] Elfekey, E.Z., Sarker, R.A., Essam, D.L.: A Simple Ranking and Selection for Constrained Evolutionary Optimization. In: Wang, T.-D., Li, X.-D., Chen, S.-H., Wang, X., Abbass, H.A., Iba, H., Chen, G.-L., Yao, X. (eds.) *SEAL 2006*. LNCS, vol. 4247, pp. 537–544. Springer, Heidelberg (2006)

Discussion of Search Strategy for Multi-objective Genetic Algorithm with Consideration of Accuracy and Broadness of Pareto Optimal Solutions

Tomoyuki Hiroyasu¹, Masashi Nishioka², Mitsunori Miki³,
and Hisatake Yokouchi¹

¹ Faculty of Life and Medical Sciences, Doshisha University,
1-3 Tatara Miyakodani Kyotanabe, Kyoto, Japan
tomo@is.doshisha.ac.jp, hyokouch@mail.doshisha.ac.jp

² Graduate School of Engineering, Doshisha University
mnishioka@mikilab.doshisha.ac.jp

³ Faculty of Science and Engineering, Doshisha University
mmiki@mail.doshisha.ac.jp

Abstract. In multi-objective optimization, it is important that the obtained solutions are high quality regarding accuracy, uniform distribution, and broadness. Of these qualities, we focused on accuracy and broadness of the solutions and proposed a search strategy. Since it is difficult to improve both convergence and broadness of the solutions at the same time in a multi-objective GA search, we considered to converge the solutions first and then broaden them in the proposed search strategy by dividing the search into two search stages. The first stage is to improve convergence of the solutions, and a reference point specified by a decision maker is adopted in this search. In the second stage, the solutions are broadened using the Distributed Cooperation Scheme. From the results of the numerical experiment, we found that the proposed search strategy is capable of deriving broader solutions than conventional multi-objective GA with equivalent accuracy.

1 Introduction

In the field of multi-objective optimization, the purpose is to find Pareto optimal solutions. To achieve this, many multi-objective genetic algorithms (MOGAs) have been developed [1, 2, 3, 4, 5]. Of the many methodologies reported to date, NSGA-II [4] and SPEA2 [5] are known to show good performance.

When searching for Pareto optimal solutions, it is important that the obtained solutions have high quality with regard to accuracy, uniform distribution, and broadness. Accuracy is how close the obtained solutions are to the true Pareto front, and uniform distribution is how evenly located the solutions are without concentrating in certain areas. Broadness is how widespread the solutions are and is decided by the solutions located at the edge of the Pareto front, which are optimal solutions of each objective.

Many MOGAs have mechanisms to improve accuracy and uniform distribution of the solutions. However, only mechanisms to store the obtained broadness of the solutions are available, with few capable of improving it. Therefore, it is difficult to improve broadness during the search, and solutions with insufficient broadness may be obtained in the case of problems with large and complex objective spaces. In addition, it is difficult to verify whether the obtained Pareto front is broad or not.

With this background, Okuda et al. proposed the Distributed Cooperation Scheme [6], which utilizes single-objective GA (SOGA) along with MOGA so that not only non-dominated solutions but optimal solutions of each objective are also searched. It was confirmed that the Distributed Cooperation Scheme is capable of deriving broader solutions than conventional MOGAs. However, preliminary experiments have also indicated that the convergence speed is reduced because the solutions are broadened from the beginning of the search.

To improve convergence or broadness of the solutions in a MOGA search, there have been a number of studies of crossover methods, such as neighborhood crossover of NCGA [7] and similarity-based mating scheme [8]. Although the purpose of our research is to improve the accuracy and broadness of the solutions, it is difficult to simultaneously improve both qualities. Therefore, we propose a search strategy composed of two search stages that separately improve convergence and broadness. The first stage in the proposed search strategy is to improve convergence of the solutions, and the second is to improve the broadness of the solutions. A reference point [9] specified by a decision maker is adopted in the first stage to accelerate convergence speed of the search. In the second stage, a Distributed Cooperation Scheme is utilized to search for optimal solutions of each objective and broaden the solutions.

2 Search Strategy for Multi-objective Genetic Algorithm with Consideration of Accuracy and Broadness

The proposed search strategy consists of two search stages as shown in Fig. 1. The first stage is a search to improve convergence, and the second is for broadness. The order of the search is set this way, as the final solutions obtained must be comparable in accuracy to conventional MOGAs and also be broad. Especially, in cases where the search time is limited, it becomes important to ensure the accuracy of the solutions first.

2.1 Convergence Search

In the convergence search, the preference of a decision maker is adopted in forms of a reference point [9]. This reference point can be located in both feasible or infeasible regions. Conventional MOGAs base their search on the dominance relationship of the solutions, but the proposed search method bases its search on the distance information. That is, solutions closer to the reference point are prioritized in the search, which leads to convergence of the solutions around the reference point. The concept of this search is illustrated in Fig. 2.

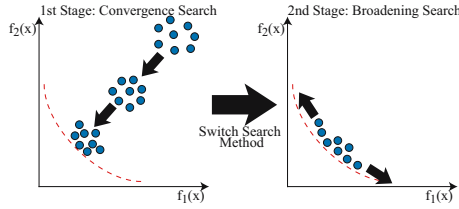


Fig. 1. Concept of Proposed Search Strategy

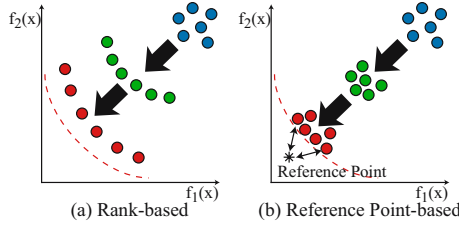


Fig. 2. Concept of Reference Point-based Search

The proposed search method is based on the conventional MOGA, and the distance information is utilized in the selection criterion of mating selection. The mating selection method is described below, and the archive size here is N .

Step 1: Sort archive solutions in ascending order of the Euclidean distance from the reference point.

Step 2: Add top $\frac{N}{2}$ solutions to the search population.

Step 3: Select remaining solutions by tournament selection based on their rank.

If multiple solutions with same rank exist, select solution with smallest Euclidean distance.

$\frac{N}{2}$ solutions close to the reference point are copied to the search population in Step 2, because these solutions are not guaranteed to be selected using methods such as tournament selection. Copying these solutions to the search population should result in improvement of convergence. In addition, both rank and Euclidean distance are considered in the tournament selection at Step 3, which allows selection of non-dominated solutions close to the reference point, and the search is directed toward the reference point while preserving diversity.

2.2 Broadening Search

The Distributed Cooperation Scheme of Okuda et al. [6] is adopted in the broadening search. The search population is divided into subpopulations that search using MOGA and SOGA in the Distributed Cooperation Scheme. Henceforth, the subpopulations searched with MOGA and SOGA are called the MOGA

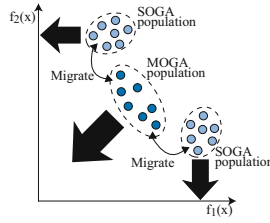


Fig. 3. Concept of Distributed Cooperation Scheme

population and SOGA population, respectively. When there are k objectives, the search population is divided into $k + 1$ subpopulations: one MOGA population and k SOGA populations. The concept of this scheme is illustrated in Fig. 3. As this is a scheme, any MOGA or SOGA methodology can be adopted. In this study, SPEA2 [5] and DGA [10] were adopted as the MOGA and SOGA populations, respectively.

MOGA and SOGA populations search in a parallel manner in the Distributed Cooperation Scheme, and best solutions from each population are exchanged every interval generations; this interval was set to 25 generations in this study. The best solution of the f_i SOGA population is the solution with the best f_i objective value. On the other hand, best solutions of the MOGA population are non-dominated solutions with best objective value for each objective, and k solutions exist in a k -objective problem. Migration of solutions in a two-objective problem is shown in Fig. 4.

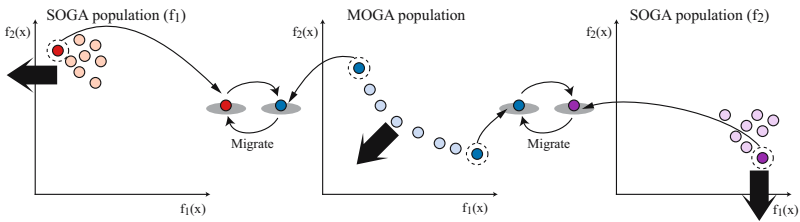


Fig. 4. Concept of Migration in Distributed Cooperation Scheme

The algorithm of the Distributed Cooperation Scheme with population size of N in the k -objective problem is shown below.

- Step 1:** Randomly generate N individuals.
- Step 2:** Divide the individuals into MOGA and k SOGA populations with $\frac{N}{k+1}$ individuals in each.
- Step 3:** Search for non-dominated solutions in the MOGA population and optimal solutions of each objective in SOGA populations.
- Step 4:** Collect solutions from all populations and update archive.

Step 5: Exchange best solutions between MOGA and SOGA populations every interval generations.

Step 6: End if terminate criterion is met, else go back to Step 3.

2.3 Search Strategy

In the proposed search strategy, the convergence search described in section 2.1 is conducted, followed by the broadening search described in section 2.2. When to switch the search stage becomes important in this case. It is preferable that the search be switched when the solutions are at nominal convergence. Therefore, we adopt a convergence indicator in switching the search stage.

The indicator utilized in MRMOGA [11] was adopted as a convergence indicator. It is an average ratio of non-dominated solutions in the archive that is dominated by the derived solutions over several generations. This ratio will be high when the search is advancing and low when converged. In detail, when non-dominated solutions of the archive at the i th generation is $PF_{known}(i)$, the ratio of $PF_{known}(i)$ that is dominated ($dominated_i$) can be calculated. Based on the average ratio over g generations, it can be determined that the search has converged if criterion (1) is met.

$$\sum_{i=1}^k \frac{dominated_i}{g} \leq \epsilon \quad (1)$$

With MRMOGA, the value of $\epsilon = 0.05$. We used $\epsilon = 0.05$ for two-objective problems and $\epsilon = 0.025$ for three-objective problems. Moreover, the period of g generations is set to be the same as the migration interval in section 2.2, which was 25 generations. The process of the search strategy for a k -objective problem is shown below.

Step 1: Initialize the archive.

Step 2: Conduct convergence search as described in section 2.1.

Step 3: Check criterion (1) every g generations. Go to Step 4 if criterion (1) is met, else go back to Step 2.

Step 4: Divide solutions stored in archive into $k + 1$ populations.

Step 5: Conduct broadening search as described in section 2.2.

Step 6: End if terminate criterion is met, else go to Step 5.

3 Numerical Experiment

A numerical experiment was performed to verify the effectiveness of the proposed search strategy by comparison with SPEA2. The MOGA methodology of the proposed search strategy is SPEA2, and DGA was adopted as the SOGA population. The test problems used in this experiment were KUR and multi-objective knapsack problems. KUR is a two-objective continuous problem with 100 design variables [12]. KP500-2 (i.e., 2 objectives, 500 items), KP750-2, and

KP750-3 [3] were selected as multi-objective knapsack problems. Lamarckian repair [13] was adopted as a repair method for the knapsack problems, and the items to be removed were selected randomly.

Many metrics are available to evaluate the obtained solutions, and we adopted inverted generational distance (IGD) [14], hypervolume (HV) [15], and spread [8]: IGD is the average distance from each solution of the Pareto optimal front to the closest obtained solution, and is a metric of accuracy and broadness; HV is a metric of overall performance; and spread, calculated as the sum of differences between maximum and minimum values of each objective within the obtained Pareto front, is a metric of broadness. The Pareto optimal front must be known to calculate IGD, but is unknown for KUR, KP750-2, and KP750-3 problems. Therefore, we obtained near Pareto optimal solutions beforehand using a much greater population size and generations, which were used in the calculations.

For both the proposed search strategy and SPEA2, population size is set to 120 and the maximum generations is 1000. Therefore, the number of evaluations is the same for both methods. In addition, 2 point crossover is utilized with crossover rate of 1.0, and the mutation rate is $1/\text{Chromosome Length}$. The parameters specific to the DGA used in the proposed search strategy are as follows: sub population size is 10, tournament selection with tournament size of 4, migration rate is 0.5, and migration interval is 5. The topology of migration is random ring.

In the proposed search strategy, a reference point must be set for each problem, and is set at $(-1000, -400)$ for KUR, $(30000, 30000)$ for KP500-2 and KP750-2, and $(30000, 30000, 30000)$ for KP750-3. These reference points were set in the area close to the center of the expected Pareto front. Further experiments on the placement of the reference points are needed in the future studies.

3.1 Results

Search results of KUR and KP750-2 by the proposed search strategy and SPEA2 in 30 trials are shown in Figs. 5 and 6.

The search results shown in Figs. 5 and 6 indicate that the proposed model obtained broader solutions than SPEA2. Broader solutions provide more information of the shape of the Pareto front, which is important especially in problems such as KUR and KP750-2 where the optimal front is unknown. In addition, the

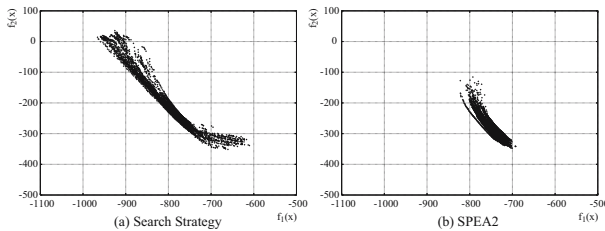


Fig. 5. Search Results of KUR (30 Trials)

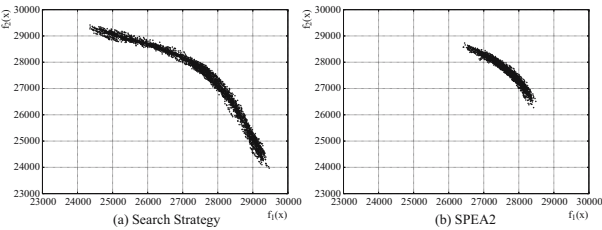


Fig. 6. Search Results of KP750-2 (30 Trials)

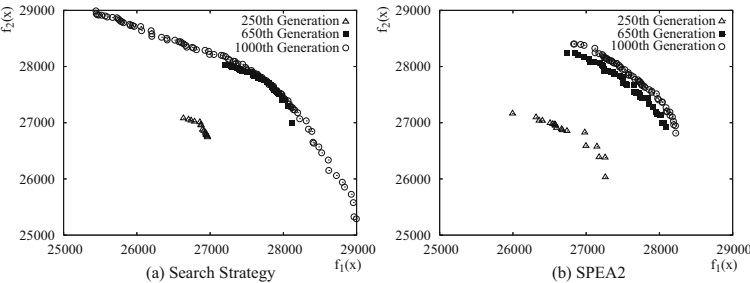


Fig. 7. Solution Set at Each Generation in a Single Run (KP750-2)

solution set of a single run for KP750-2 is shown in Fig. 7. With the search strategy, the search was switched from convergence search to broadening search at the 650th generation as shown in Fig. 7(a).

Fig. 7(a) shows that the search strategy is successful in first converging and then broadening solutions, as the broadness of the converged solutions improved after switching to the broadening search. On the other hand, solutions of SPEA2 shown in Fig. 7(b) are not broadened much as the search progresses. Similar results were also observed in other problems.

Second, the mean values and standard deviation of IGD, spread, and HV values are shown in Tables 1 to 3. For IGD in Table 1, the obtained solutions were closer to the Pareto optimal front when the value was close to 0. On the other hand, solutions with greater values of spread and HV are better.

Table 1. Inverted Generational Distance

	KUR	KP500-2	KP750-2	KP750-3
Search Strategy: mean	0.08343	0.01498	0.01767	0.03443
SD	0.04859	0.00159	0.00236	0.00233
SPEA2: mean	0.13593	0.02966	0.03086	0.04949
SD	0.01889	0.00198	0.00195	0.00287

Table 2. Spread

	KUR	KP500-2	KP750-2	KP750-3
Search Strategy: mean	516.22	6271.06	8952.60	6271.06
SD	193.42	355.85	648.39	1455.15
SPEA2: mean	243.62	2545.80	3122.00	2519.06
SD	23.04	264.68	295.57	282.89

Table 3. Hypervolume

	KUR	KP500-2	KP750-2	KP750-3
Search Strategy: mean	2.635E+05	3.919E+08	8.461E+08	2.408E+13
SD	18832.6	1.652E+06	4.546E+06	5.425E+11
SPEA2: mean	2.550E+05	3.705E+08	8.013E+08	2.263E+13
SD	7563.9	1.445E+06	3.555E+06	1.233E+11

From the mean values of IGD in Table 1, it can be seen that the proposed model is equivalent to or better than SPEA2. Therefore, the proposed model is comparable to SPEA2 with regard to accuracy. In addition, IGD also indicates how close the obtained solutions are to the optimal front regarding broadness. Therefore, the obtained solutions of SPEA2 are not sufficiently broad. The spread values shown in Table 2 also indicate that the proposed model obtained broader solutions. Therefore, the approach to broaden solutions after converging the search is capable of obtaining broad solutions. Mean HV values shown in Table 3 also show better results for the search strategy. Next, the transition of mean IGD values and mean spread values of KUR and KP750-2 are shown in Figs. 8 and 9, respectively, to verify whether the targeted search is achieved.

The mean IGD values in Fig. 8 are better when the obtained solutions are close to the true Pareto front and broad. With the search strategy, the search was switched at means of 835 and 543 generations for KUR and KP750-2, respectively. As shown in Fig. 8, IGD values of the search strategy improved

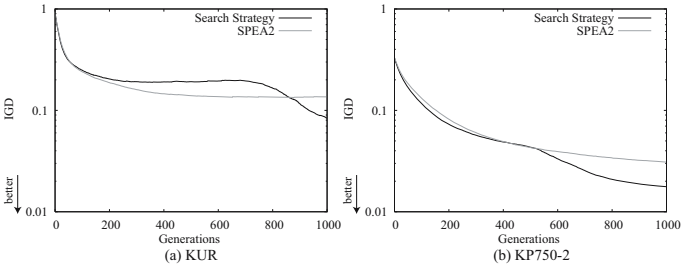


Fig. 8. Transition of Mean IGD Values

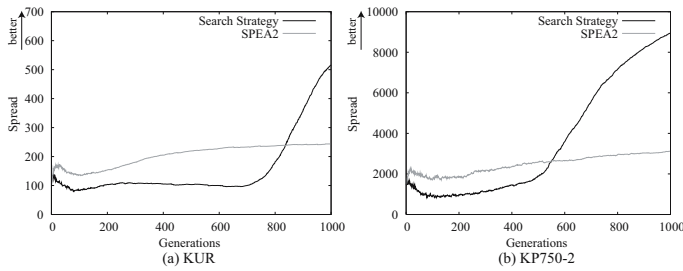


Fig. 9. Transition of Mean Spread Values

greatly after the search was switched. In addition, the spread values of the search strategy in Fig. 9 also improved after the search was switched. These results confirmed that the solutions converged and then broadened in the search strategy, and that the targeted search was achieved.

These results indicated that the proposed search strategy is effective for maintaining accuracy comparable to conventional MOGAs and deriving broader solutions. However, the variance of the performance by the proposed search strategy was greater than that of SPEA2. In some trials, the search was switched from the first to the second stage late in the search, because it took many generations to reach nominal convergence. This resulted in insufficient broadness of the solutions. Therefore, further studies to determine how to switch the search are required.

4 Conclusions

In this paper, we focused on the accuracy and broadness of the solutions and proposed a search strategy for MOGAs. As it is difficult to improve both convergence and broadness of the solutions at the same time in a MOGA search, we considered converging the solutions first and then broadening them in the proposed search strategy. To accomplish this, the search is divided into two search stages. The first stage improves the convergence of the solutions, and a reference point specified by a decision maker is adopted for this purpose. In the second stage, the solutions are broadened using the Distributed Cooperation Scheme.

The results of numerical experiments indicated that the proposed search strategy can derive broader solutions compared to conventional MOGA with comparable accuracy. In future studies, a mechanism to judge when to switch the search stage must be determined, as it is now controlled by a parameter, and performance of the search strategy is dependent on this issue.

References

- [1] Goldberg, D.E.: Genetic Algorithms in search, optimization and machine learning. Addison-Wesley (1989)
- [2] Fonseca, C.M., Fleming, P.J.: Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In: Proceedings of the 5th international conference on genetic algorithms, pp. 416–423 (1993)
- [3] Zitzler, E., Thiele, L.: Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation* 3(4), 257–271 (1999)
- [4] Deb, K., Agarwal, S., Pratap, A., Meyarivan, T.: A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. In: KANGAL report 200001, Indian Institute of Technology, Kanpur (2000)
- [5] Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the Performance of the Strength Pareto Evolutionary Algorithm. In Technical Report 103, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH) Zurich (2001)
- [6] Okuda, T., Hiroyasu, T., Miki, M., Watanabe, S.: DCMOGA: Distributed Co-operation model of Multi-Objective Genetic Algorithm. In: Advances in Nature-Inspired Computation: The PPSN VII Workshops, pp. 25–26 (2002)
- [7] Watanabe, S., Hiroyasu, T., Miki, M.: NCGA: Neighborhood Cultivation Genetic Algorithm for Multi-Objective Optimization Problems. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002), pp. 458–465 (2002)
- [8] Ishibuchi, H., Shibata, Y.: Mating Scheme for Controlling the Diversity-Convergence Balance for Multiobjective Optimization. In: Deb, K., et al. (eds.) GECCO 2004. LNCS, vol. 3102, pp. 1259–1271. Springer, Heidelberg (2004)
- [9] Deb, K., Sundar, J.: Reference Point Based Multi-Objective Optimization Using Evolutionary Algorithms. In: GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation, pp. 635–642 (2006)
- [10] Tanese, R.: Distributed Genetic Algorithms. In: Proc. 3rd ICGA, pp. 434–439 (1989)
- [11] Jaimes, A.L., Coello, C.A.C.: MRMOGA: Parallel Evolutionary Multiobjective Optimization using Multiple Resolutions. In: 2005 IEEE Congress on Evolutionary Computation (CEC 2005), pp. 2294–2301 (2005)
- [12] Kursawe, F.: A Variant of Evolution Strategies for Vector Optimization. In: Parallel Problem Solving from Nature. 1st Workshop, PPSN I, pp. 193–197 (1991)
- [13] Ishibuchi, H., Kaige, S., Narukawa, K.: Comparison between Lamarckian and Baldwinian Repair on Multiobjective 0/1 Knapsack Problems. In: Coello Coello, C.A., Hernández Aguirre, A., Zitzler, E. (eds.) EMO 2005. LNCS, vol. 3410, pp. 370–385. Springer, Heidelberg (2005)
- [14] Sato, H., Aguirre, H., Tanaka, K.: Local Dominance Using Polar Coordinates to Enhance Multi-objective Evolutionary Algorithms. In: Proc. 2004 IEEE Congress on Evolutionary Computation, pp. 188–195 (2004)
- [15] Knowles, J., Thiele, L., Zitzler, E.: A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers. In TIK Report 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich (2006)

Discussion of Offspring Generation Method for Interactive Genetic Algorithms with Consideration of Multimodal Preference

Fuyuko Ito¹, Tomoyuki Hiroyasu², Mitsunori Miki³, and Hisatake Yokouchi²

¹ Graduate School of Engineering, Doshisha University,
1-3 Tatara Miyakodani Kyotanabe, Kyoto, Japan
fuyuko@mikilab.doshisha.ac.jp

² Department of Life and Medical Sciences, Doshisha University
tomo@is.doshisha.ac.jp, hyokouch@mail.doshisha.ac.jp

³ Department of Science and Engineering, Doshisha University
mmiki@mail.doshisha.ac.jp

Abstract. The interactive genetic algorithm(iGA) is a method to obtain and predict a user's preference based on subjective evaluation of users, and it has been applied to many unimodal problems, such as designing clothes or fitting of hearing aids. On the other hand, we are interested in applying iGA to user's preferences, which can be described as a multimodal problem with equivalent fitness values at the peaks. For example, when iGA is applied to product recommendation on shopping sites, users have several types of preference trends at the same time in product selection. Hence, reflecting all the trends in product presentation leads to increased sales and consumer satisfaction. In this paper, we propose a new offspring generation method that enables efficient search even with multimodal user preferences by introducing clustering of selected individuals and generating offspring from each cluster. Furthermore, we perform a subjective experiment using an experimental iGA system for product recommendation to verify the efficiency of the proposed method. The results confirms that the proposed method enables offspring generation with consideration of multimodal preferences, and there is no negative influence on the performance of preference prediction by iGA.

1 Introduction

The interactive genetic algorithm(iGA)[1] is an optimization method in which users evaluate the solutions instead of the fitness function of a genetic algorithm (GA)[2] and it optimizes targets that could not be formulated by iterating the following steps: presentation of individuals to users, evaluation of individuals by users, selection, crossover, and mutation. However, there are a number of problems in iGA, such as prediction of fitness values, combating user fatigue[3], etc. For example, prediction of fitness model of user's preference accelerates the iGA convergence and reduces user fatigue. In this study, we focus on prediction of fitness model or user's preference and how to make progress in presenting population to users.

As iGA replaces the fitness function with a user's preference, the landscape of preference differs widely depending on the target problems of iGA. If there are some trends in a user's preference, the landscape is multimodal with equivalent fitness values at the peaks and the problem has multiple optimum solutions. However, conventional iGA searches for only one of these trends. Toward the end of such a search, the population converges to an optimal solution and very few individuals that are similar to other optimal solutions are presented. Particularly, for multimodal function with equivalent fitness values at the peaks, it is preferable to present all peaks to users during the search.

Applying iGA to product recommendation on shopping sites is one such problem. Product recommendation, such as collaborative filtering(CF)[4,5], is a technique utilized to present products that are likely to be bought based on a user's preferences. When users select products on shopping sites, there are several types of preference trends at the same time, which can be described as multimodal functions with equivalent fitness values at the peaks.

Here, we discuss an offspring generation method of iGA that enables efficient search even if user's preference is described as a multimodal function with equivalent fitness values at the peaks. The proposed method introduces population clustering to an iGA. The following section presents an application of iGA for product recommendation and associated problems. Then, the proposed offspring generation method considering multimodal preferences is described. Finally, the effectiveness of the proposed method is verified by experiment with a recommendation system using iGA.

2 Interactive Genetic Algorithms and Multimodal Preference

2.1 Interactive Genetic Algorithms

The interactive genetic algorithm (iGA)[1] is an optimization method based on subjective evaluation of users and the genetic operations of GA[2]. Subjective preference replaces the fitness function for evaluation operation of GA in iGA. Therefore, iGA has been applied as a method of sensitivity analysis to many problems that are difficult to formulate, such as the design of clothes and music composition. Aoki et al.[6] also confirmed that iGA is especially effective for users inexperienced with the target problem. IGA consists of the following steps:

1. Generate an initial population of N_{pop} individuals.
2. Present the individuals to the user.
3. Evaluate each individual based on the user's preference.
4. Select individuals to be kept for the next generation from the user's evaluation.
5. Exchange chromosomes between individuals and generate offspring.

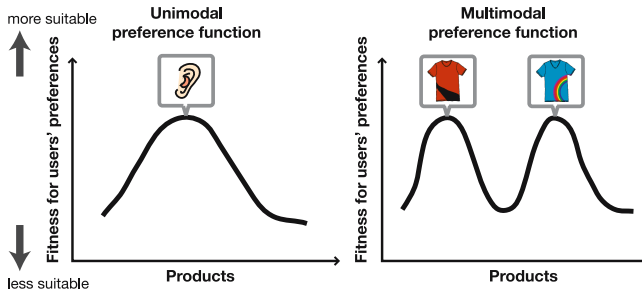


Fig. 1. Unimodal and multimodal preference functions

6. Randomly change chromosomes to maintain diversity.
7. Terminate if the user obtains desired individual, else iterate steps 2 to 6.

On the other hand, a number of daunting challenges of iGA remain: discrete fitness value input method, prediction of fitness values, interface for dynamic tasks, acceleration of iGA convergence, combination of evolutionary and non-evolutionary computation, active intervention, and theoretical research. In this study, we focus on prediction of fitness model of user's preference and examine an offspring generation method to make progress on presenting population to users with multimodal preferences.

2.2 IGA for Multimodal Preference

When iGA is applied to actual problems, the landscape of the fitness function, the user preferences in this case, is very important. For example, the problems of parameter settings of a hearing aid can be transferred into interactive optimization problems. In this case, the best combination of parameters that will help the user to hear may exist. Thus, landscape of preferences of the hearing aid can be described as shown in Fig.1. In this figure, a single optimal solution should be selected from individuals in a unique optimal region. On the other hand, when users select products at shopping sites, each user generally has several types of preferences. Therefore, the landscape of the preferences on product selection can be described as a multimodal function, as shown in Fig.1. In this case, the multiple optimal solutions should be selected from individuals in all optimal regions. Hence, it is better to present individuals in all optimal regions corresponding to each preference.

However, iGA searches for a single optimal solution and it is not beneficial for users if all individuals converged to only a single optimal region of multimodal preference. Therefore, an offspring generation method that responds to not only unimodal preferences, but also multimodal preferences is necessary. To achieve this, a new offspring method using clustering is proposed in this paper.

3 Offspring Generation Method with Consideration of Multimodal Preference

3.1 Overview of the Proposed Method and Its Execution Timing

To enable offspring generation considering multimodal preferences, each peak of preference function must be discovered. To obtain local optima of a multimodal function, Hocaoglu and Sanderson[7] introduced clustering to GA, and the population is divided into several sub-populations. These sub-populations evolve independently, exchanging individuals, and are merged at a certain interval. Moreover, sharing (or niching) reveals the distribution of local optima and maintains the diversity of the population by updating the fitness value of each individual based on degree of congestion. According to the former, we assume that each peak of multimodal preferences corresponds to scattered regions in the design domain and propose an offspring generation method taking multimodality into consideration using clustering of highly evaluated individuals.

In the proposed method, a user clicks favorite individuals and each individual is evaluated with regard to whether the user has clicked it or not. The regions that suit the user's preference are specified from the design space by clustering, and offspring are generated from these regions. The number of clusters must be determined automatically and the number of regions is not known initially. Hence, a clustering method which automatically determines the number of clusters is adopted in the proposed method. Moreover, the proposed offspring generation method is applied once at the m th generation of iGA, as shown in Fig.2. In other generations, conventional crossover and mutation operators are utilized to generate offspring. Thus, it is possible to generate offspring that are suitable to the user's preference beyond specified regions.

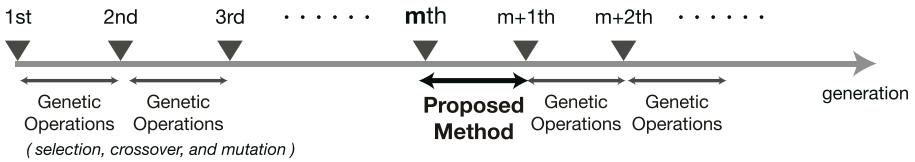


Fig. 2. Application timing of the proposed method for iGA

3.2 Procedure of Offspring Generation with Consideration of Multimodal Preference

Figure 3 shows an overview of the proposed method. In an n -dimensional design space, the proposed method generates N_{off} offspring as follows:

1. At the m th generation, all individuals selected by the users up to m th generation are divided into clusters, $C_1, \dots, C_k, \dots, C_{N_{cluster}}$ in the design space, with the number of clusters $N_{cluster}$ is determined automatically.

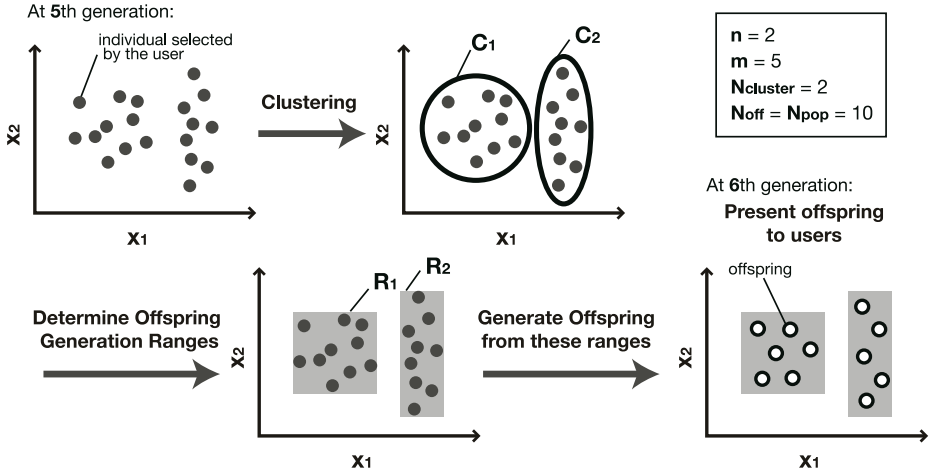


Fig. 3. Offspring generation procedure in the proposed method

2. The offspring generation range R_k corresponding to cluster C_k is determined by the distribution range of individuals $I_{k1}, I_{k2}, \dots, I_{kl}$ in C_k .
3. N_{off} offspring (as many as population size N_{pop}) are generated randomly from offspring generation ranges $R_0, \dots, R_{N_{cluster}}$. Each range generates the same number of offspring.
4. Generated offspring are presented at the $m + 1$ th generation and operations of conventional iGA are iterated subsequently.

As described above, the proposed method specifies the regions that suit the user's preferences from the distribution range of the individuals selected by the user up to the m th generation. These individuals are divided appropriately into clusters and it enables offspring generation with consideration of not only uni-modal but also multimodal preferences.

4 Assessment of Offspring Generation with Consideration of Multimodal Preference

4.1 Experimental iGA System for Product Recommendation

For assessment of the proposed offspring generation method, we develop a product recommendation system using iGA and adopted t-shirts as the target product. The users evaluate the presented products by clicking based on whether they are suitable for their preferences. Design variables of a t-shirt and the flow of the experimental system are shown below:

Design Variables. The design variables of a t-shirt consist of color, shape, and pattern. Expressions of each design variable are shown in Fig.4. The color of a t-shirt is expressed by the HSB color system, which is similar to the human sense of

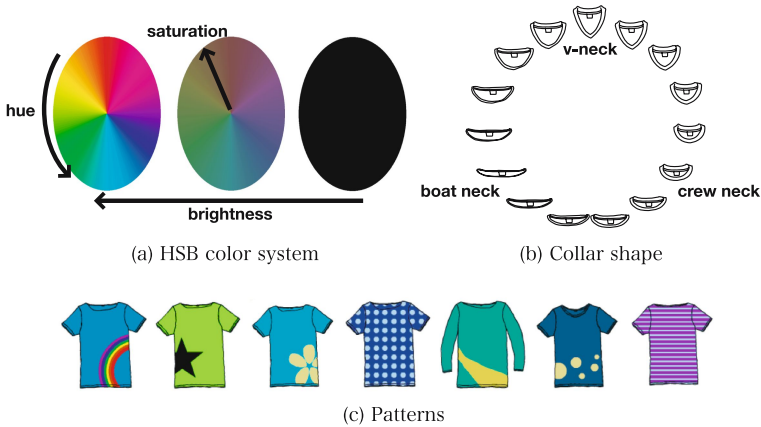


Fig. 4. Design variables of a t-shirt

color recognition. The HSB color system expresses color as a combination of hue (0 to 360 degrees), saturation (0 to 100), and brightness (0 to 100) as shown in Fig.4(a). These design variables of the HSB color system are continuous values. The shape of a t-shirt is described by the shape of the collar and the length of the sleeves as discrete values. Fifteen collar shapes are constructed based on three basic shapes—boat neck, v-neck, and crew neck—as shown in Fig.4(b). The sleeve types are short sleeves and long sleeves. Eight t-shirt patterns including a solid color are shown in Fig.4(c), and the color of each pattern is determined from 9 colors: white, yellow-green, sky blue, blue, purple, pink, red, yellow, and black. The design variables of t-shirt patterns and their colors are discrete values.

Flow of Experimental System

– Generation and presentation of the initial population

First, the system generates N_{pop} individuals as an initial population. The design variables of the initial population are determined randomly. Saturation and brightness are determined from the range of 75 to 100 to present a wide range of hue clearly.

– Evaluation and selection

Users click the products on the screen that suit their preferences. Presented individuals are evaluated by whether they are clicked or not. Individuals clicked by the user are selected as parents. The number of parents is set to half the population size N_{pop} , and users are allowed to click up to half of N_{pop} individuals. If clicked individuals are less than the number of parents, the system automatically selects the remaining parent individuals from the population in increasing order of Euclidean distance from the clicked individuals.

- Crossover
The crossover operator generates two offspring from two parents. Hue and collar shape of the offspring are determined from the acute angle made by two parents in the circular design space. Saturation and brightness of offspring are determined between the design variables of two parents in the design space. Other design variables of offspring are inherited from the parents.
- Mutation
The mutation operator changes the design variables of offspring at random based on mutation rate.

4.2 Experimental Overview

An experiment is performed to determine whether the proposed method generates offspring with consideration of multimodal preferences. In this experiment, we adopt the experimental product recommendation system described in Section 4.1. Subjects utilize the system with the proposed offspring generation method as the proposed system and the system with only the conventional genetic operations as the conventional system. Each system shows 20 individuals in one generation and presents t-shirts in the interface, as shown in Fig.5. The subjects consist of 14 men and 6 women ranging in age from 21 to 27 years, and the order of the systems used is counterbalanced among subjects. They are instructed to select their favorite t-shirts in each system and to evaluate the t-shirts presented in each system for 10 generations. The crossover rate and mutation rate are set to 1.0 and 0.2, respectively. Multiobjective clustering with automatic k-determination (MOCK)[8] which automatically determines the number of clusters is adopted to generate offspring. The proposed system generates offspring using clustering after evaluation of the 5th generation and presents the generated individuals at the 6th generation. Hue and collar shape are adopted as the target design variables for clustering, and the number of clusters is set in the range of 1 to 3 in this experiment. Subjects answer questionnaires after experimenting with each system.

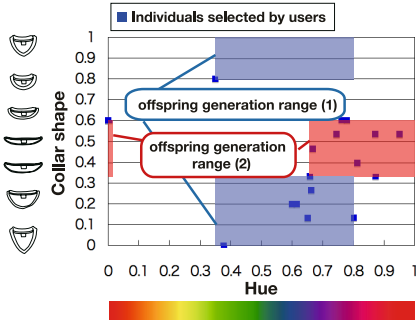


Fig. 5. The experimental system interface

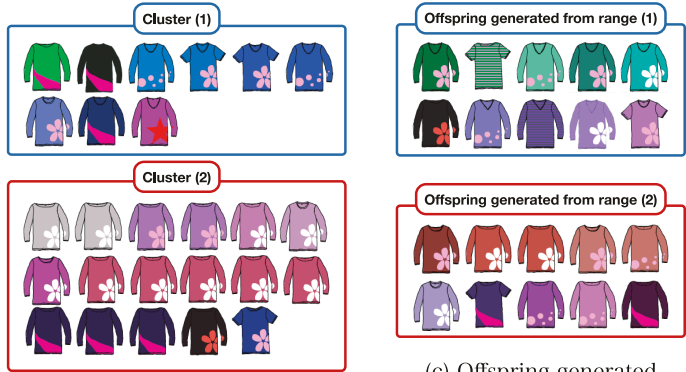
4.3 Results and Discussion

First, we discuss whether the proposed method generated offspring that considered multimodal preferences or not. Figure 6(a) shows the offspring generation ranges of subject A determined by the proposed method and the individuals selected by the user up to the 5th generation before applying the proposed method. The horizontal and vertical axes show hue and collar shape, respectively. The offspring generation range (1) seems to be divided horizontally, and range (2) also seems to be divided vertically. However, they are not divided because the design space was circular. The two ranges overlaps each other around the range of purple, but selected individuals are divided into two clusters (see Fig.6).

Next, Fig.6(b) shows the results of clustering of selected individuals. Clusters (1) and (2) consist of cool and warm color t-shirts, respectively. Therefore, subject A has multimodal preferences and liked both cool and warm colors. Cluster (1) also consists of v-neck t-shirts, while cluster (2) consists of boat neck t-shirts. Black and flower-patterned t-shirts are included in both clusters because brightness and patterns of t-shirts are not adopted in clustering in this experiment. Thus, target design variables for clustering should be discussed in future studies.



(a) Selected individuals and offspring generation ranges



(b) Clustering result

(c) Offspring generated by the proposed method

Fig. 6. Example of offspring generation considering multimodal preferences

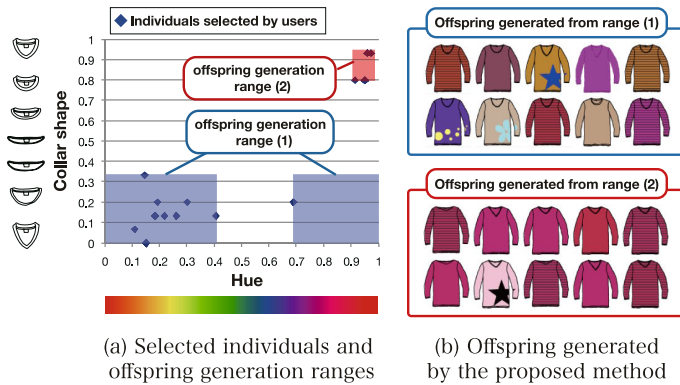


Fig. 7. Example of failure by the proposed method

Moreover, Fig.6(c) shows the list of individuals presented at the 6th generation that are generated from these offspring generation ranges. Purple t-shirts are generated from both ranges because they overlapped with each other (see Fig.6(a)). However, the population did not converge on one of the user's multiple preferences. These trends of experimental results are similar to 12 of 20 subjects and thus the proposed method would be able to generate offspring considering multimodal preferences.

In addition, a case where the proposed method do not perform efficiently is also discussed. Figure 7(a) shows the offspring generation ranges and the individuals that are selected by subject B up to the 5th generation. Figure 7(b) shows the offspring generated from these ranges. Subject B selects many green t-shirts by the 5th generation (see Fig.7(a)) and one of the green t-shirts is selected five times. However, no green t-shirts are included in the offspring generated by the proposed method (see Fig.7(b)). There is a possibility that green t-shirts would be generated from the offspring generation range (1), but it is not because the offspring were generated without considering the distribution of selected individuals in each range. To achieve this, offspring generation with a probabilistic model should be discussed in future studies.

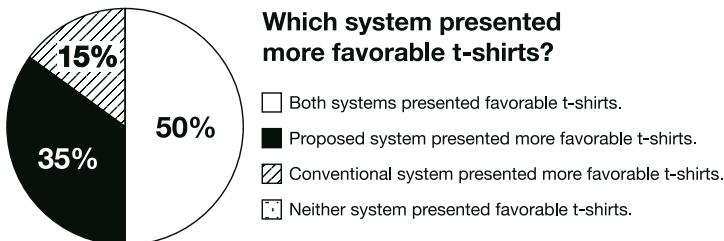


Fig. 8. Questionnaire results

A questionnaire study is used to assess whether the proposed offspring generation method has negative effects on the performance of preference learning of iGA. The question is, gWhich system presented more favorable t-shirts?h The subjects choose their responses from among 4 choices as shown in Fig.8. Eighty-five percent of the subjects respond that the proposed system or both systems presented individuals that suit their preferences (see Fig.8). Thus, subjective evaluation confirms that the proposed offspring generation method using clustering did not negatively affect the performance of preference learning of the iGA.

5 Conclusions and Future Works

The iGA is an effective method to introduce a user's preference to a system. It is capable of converging on unique optima by conventional iGA when a user's preference is unimodal. On the other hand, a mechanism that enables efficient search even if a user's preference is multimodal with equivalent fitness values at the peaks is needed. Therefore, we propose a new offspring generation method considering multimodal preferences. The proposed method introduces clustering of selected individuals to obtain local-optima and generates offspring from each cluster. We performed a subjective experiment using an experimental iGA system for product recommendation. The experimental results indicate that the proposed method could appropriately determine trends in a user's preference and generate offspring with consideration of multimodal preferences. In future work, we will discuss the target design variables for clustering and offspring generation with a probabilistic model from specified regions considering dependencies among design variables. Furthermore, we will also improve the method of specification of regions that suit a user's preference.

References

1. Takagi, H.: Interactive evolutionary computation: Fusion of the capabilities of ec optimization and human evaluation. *Proceedings of the IEEE* 89(9), 1275–1296 (2001)
2. Goldberg, D.E.: *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, Reading (1989)
3. Llorà, X., Sastry, K., Goldberg, D.E., Gupta, A., Lakshmi, L.: Combating user fatigue in igas: partial ordering, support vector machines, and synthetic fitness. In: *Proceedings of Genetic Evolutionary Computation Conference (GECCO)*, pp. 1363–1370 (2005)
4. Konstan, J.A., Miller, B.N., Maltz, D., Herlocker, J.L., Gordon, L.R., Riedl, J.: GroupLens: applying collaborative filtering to usenet news. *Commun. ACM* 40(3), 77–87 (1997)
5. Sarwar, B., Karypis, G., Konstan, J., Reidl, J.: Item-based collaborative filtering recommendation algorithms. In: *WWW 2001: Proceedings of the 10th international conference on World Wide Web*, pp. 285–295. ACM, New York (2001)

6. Aoki, K., Takagi, H.: Interactive ga-based design support system for lighting design in 3-d computer graphics. *The transactions of the Institute of Electronics, Information and Communication Engineers* 81(7), 1601–1608 (1998)
7. Hocaoglu, C., Sanderson, A.C.: Multimodal function optimization using minimal representation size clustering and its application to planning multipaths. *Evolutionary Computation* 5(1), 81–104 (1997)
8. Handl, J., Knowles, J.: Improving the scalability of multiobjective clustering. In: *Proceedings of the Congress on Evolutionary Computation (CEC 2005)*, vol. 3, pp. 2372–2379 (2005)

Solving Very Difficult Japanese Puzzles with a Hybrid Evolutionary-Logic Algorithm

E.G. Ortiz-García¹, Sancho Salcedo-Sanz^{1,*}, A.M. Pérez-Bellido¹,
A. Portilla-Figueras¹, and X. Yao²

¹ Department of Signal Theory and Communications, Universidad de Alcalá,
Alcalá de Henares, Madrid, Spain

² The Centre for Research in Computational Intelligence and Applications
(CERCIA), School of Computer Science, The University of Birmingham,
Birmingham, U.K. and Nature Inspired Computation and Applications Laboratory
(NICAL), University of Science and Technology of China, Hefei, P.R. China
`sancho.salcedo@uah.es`

Abstract. In this paper we present a hybrid evolutionary algorithm to solve a popular logic-type puzzle, the so called Japanese puzzle. We propose to use the evolutionary algorithm in order to initialize a logic ad-hoc algorithm, which works as a local search and implicitly defines the fitness function of the problem. Two novel operators, one for initializing the evolutionary algorithm and a second one providing a novel type of mutation adapted to Japanese puzzles are described in the paper.

1 Introduction

This paper presents a hybrid evolutionary algorithm to solve very difficult versions of a popular type of puzzles known as *Japanese puzzles*, which have gained popularity in the last few years all around the world. A Japanese puzzle is a interesting and addictive game, which take the form of a $N \times M$ grid, with numbers situated on the left and the top rows and columns. There are two types of Japanese puzzles, black and white and color puzzles. In black and white Japanese puzzles, the numbers in rows and columns represent how many blocks of cells must be filled in the grid, in the order they appear. If there are two or more numbers, the blocks of cells in the grid must be separated by at least one blank square.

For example, Figure 1 (b) shows the solution to the Japanese puzzle displayed in Figure 1 (a). Note that in the first row a 6 cells block must be filled, whereas in the second row two blocks of 3 cells each must be filled, with at least one blank between them. Columns in the puzzle follow the same rules. When all constraints in rows and columns are fulfilled, the puzzle is solved, and a nice

* This work has been partially supported by a CAM-UAH grant number CCG07-UAH/TIC-1894 and by Ministerio de Educacin y Ciencia under grant number TEC2006/7010.A. M. Pérez-Bellido and E. G. Ortiz-García are supported by Universidad de Alcalá, under the University F.P.I. grants program.

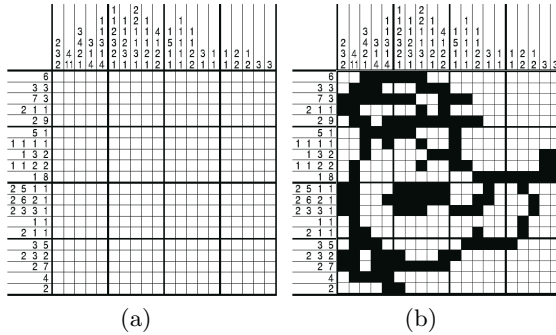


Fig. 1. Example of a Japanese puzzle; (a) Grid with conditions; (b) Solution

picture can be seen, in this case a famous picture of Popeye. See [1], [2] and [3] for more examples on Japanese puzzles.

Japanese puzzles can be used as test bed problems in the development of new computational algorithms [4], [5], also applicable to the solution and generation of the puzzles [6], [7]. Regarding the algorithms for solving Japanese puzzles (solvers), there are two main types of approaches which can be applied: logic-type algorithms and metaheuristics approaches. Logic-type solvers use local information in the grid and the cells already fixed in order to construct a solution for the puzzle from scratch [6]. Metaheuristics approaches usually try to construct a complete solution for the puzzle using the information given by an objective function which measures the goodness of the different solutions encoded by the algorithm [8], [10]. In general, logic-type solvers are more adequate for obtaining fast solutions to the puzzles than metaheuristics, which have problems of scalability in these puzzles. On the other hand, logic-type puzzles may have problems when an initial set of fixed cells cannot be found. Usually, very difficult Japanese puzzles are constructed in such a way that this happens [2].

In this paper we propose a hybrid evolutionary-logic algorithm to solve any type of black and white Japanese puzzles, including those which can be classified as very “difficult” puzzles. Our evolutionary algorithm encodes the initial state of several cells used by a logic-type solver. It then obtains the fitness value of each individual by running the logic solver and obtaining a measure of how long is the solver’s solution from the optimal solution of the puzzle. In order to improve the performance of our algorithm we propose a special initialization of the evolutionary algorithm and also a novel mutation operator specific for solving Japanese puzzles. Both novelties are based on the calculation of the a-priori probabilities of bits in a Japanese puzzle, also studied in this paper. Several experiments in 16 puzzles classified as “very difficult” in a well known web page [2], devoted to Japanese puzzles show the very good properties of the algorithm presented in this work.

The rest of the paper is structured as follows: next section briefly describe several previous approaches that can be found in the literature about solvers. Section 3 describes the hybrid evolutionary algorithm proposed in this paper, including the calculation of a-priori probabilities of bits in a Japanese puzzle and the structure of the algorithm. Section 4 shows the performance of our evolutionary algorithm in a set of very difficult Japanese puzzle's instances. Section 5 closes de paper giving some final remarks.

2 Previous Approaches

It is possible to classify the algorithms to solve Japanese puzzles into two main groups, *objective function guided* algorithms and *logic-type* algorithms. The objective function guided approaches include metaheuristics approaches and emergent solutions, such as genetic or evolutionary algorithms.

There have been several attempts to use genetic algorithms (GAs) to solve picture-logic puzzles. The first of such attempts can be found in [8], where a solver based on a GA with operators based on discrete tomography was presented. Another paper dealing with GA for solving Japanese puzzles can be found in [9]. In that paper, a comparison between a GA and a depth first search algorithm for Japanese puzzles is presented. The performance of a hybrid evolutionary algorithm for Japanese puzzles, and its application in education has been recently explored in [10]. In that work, an EA with a special encoding based on *alleles* (groups of bits which form a row or column) is proposed. In general, GAs are able to find solutions to small or medium size picture-logic puzzles, basing only on the information provided by the objective function, without considering the local information of the puzzle. Note that objective function guided algorithms usually evaluates full solutions to the puzzle.

The opposite situation appears when a logic-type algorithm is used. These algorithms do not evaluate full solutions to the puzzle, but they construct a solution from scratch, using the local information of the puzzle to fix cells to 1 or 0 (black or white). Several logic solvers have been proposed in the literature in the last few years [11], [12]. In [6], a powerful solver based on an ad-hoc algorithm, called *logic ad-hoc*, for solving Japanese puzzles has been proposed. This algorithm can also be applied to color Japanese puzzles and has been used to generate puzzles in a efficient way [7]. The logic ad-hoc algorithm works with a set of *unknowns* (cells which are not yet fixed as being filled or left blank. They are marked in grey in the figure), and progresses by fixing cells to be filled or blank, by means of a set of logic procedures (see reference [6] for details on this solver). The logic ad-hoc algorithm exclusively exploits the logic information of the puzzle, so no objective function is needed in this case.

There are some difficult puzzles in which the logic information of the puzzle is scarce, so the algorithms such as the logic ad-hoc approach do not work, or it take a lot of time to solve the puzzle. In this paper we focus on these kind of puzzles, and propose a hybrid evolutionary algorithm which tackles this point.

3 Hybrid Evolutionary Algorithm Proposed

In this paper we present a hybrid evolutionary algorithm, where the local search is the logic ad-hoc algorithm in [6]. The evolutionary algorithm encodes the puzzle using a ternary encoding, similar to the one used in the logic ad-hoc heuristic: 0 for cells in white, 1 for cells in black and 2 for unknown cells. The fitness function is a measure of the degree of puzzle solving when the logic ad-hoc algorithm is applied. It is easy to see that the idea of this algorithm is quite different to the existing genetic algorithms for Japanese puzzles solving: in this case, each individual of the evolutionary algorithm is a possible initialization of the logic ad-hoc heuristic, and we try to find the optimal initialization, which solves the puzzle.

In the next subsection we present the details of the evolutionary algorithm used, which includes two adaptations (a novel initialization and a special mutation operator) coming from the Japanese puzzles we tackle. In order to describe these novelties, we show that it is possible to obtain the a-priori probabilities of bits in a Japanese puzzle, only considering the puzzle grid and conditions in row and columns.

3.1 Calculation of A-Priori Probabilities of Bits in Japanese Puzzles

Let us consider a Japanese puzzle, with N rows and M columns, where we also have the following known parameters:

- K_i : number of conditions in a given row.
- K_j : number of conditions in a given column.
- c_{ik} : k -th condition of row i , $i = 1, \dots, N$.
- c_{jk} : k -th condition of column j , $j = 1, \dots, M$.

We look for the calculation of the a-priori probability that a given cell will be 1 in the puzzle. For carry out this, we first calculate the total number of feasible cell blocks distributions, in each row or column of the puzzle, given the initial puzzle's parameters shown above. For a given row, this number can be written as:

$$RT_i = \binom{r_i + n_i - 1}{r_i} \quad (1)$$

where $n_i = K_i + 1$ is the number of cell blocks plus one, and

$$r_i = M - \sum_{k=1}^{K_i} c_{ik} - K_i + 1, \quad (2)$$

is the total number of blank cells situated between each two cell blocks.

In the same way, for a given column, this number is:

$$CT_j = \binom{r_j + n_j - 1}{r_j}, \quad (3)$$

with $n_j = K_j + 1$, and

$$r_j = N - \sum_{k=1}^{K_j} c_{jk} - K_j + 1, \quad (4)$$

Note that these calculations are based on the number of combinations with repetitions, where n_i is the number of elements from which you can choose and r_i is the number to be chosen. Next, we also need to obtain the total number of feasible cell blocks distributions in row i and column j , considering that the cell (i, j) is fixed to be black. For the row case, this number will be equal to the sum of possible cell blocks distribution with each condition c_{ik} situated over the cell (i, j) in all possible positions (moving the condition m cells from the left-most position).

For each condition c_{ik} and each m , we can divide the row in two segments. Each segment can be seen as a equivalent row with a group of conditions to place. The first segment contains those conditions previous to the condition c_{ik} , i.e. c_{ir} with $r = \{1 \dots k-1\}$, and the second one contains those conditions next to the condition c_{ik} . Thus, we can calculate the feasible cell blocks distributions in each segment, using Equation (1). The product of these values and its sum is the number of possible cells blocks distributions wanted:

$$R_{ij} = \sum_{k=1}^{K_i} \sum_{m=1}^{c_{ik}} \binom{(L1)_{ijkm} - (s1)_{ik} + k - 1}{(L1)_{ijkm} - (s1)_{ik}} \binom{(L2)_{ijkm} - (s2)_{ik} + K_i - k}{(L2)_{ijkm} - (s2)_{ik}} \quad (5)$$

where

$$(L1)_{ijkm} = j - c_{ik} + m - 2, \quad (6)$$

is the length of the first segment given a condition c_{ik} and its relative position m ,

$$(L2)_{ijkm} = M - j - m, \quad (7)$$

is the length of the second segment given a condition c_{ik} and its relative position m ,

$$(s1)_{ik} = k - 2 + \sum_{r=1}^{k-1} c_{ir} \quad (8)$$

is the sum of conditions and fixed blank squares in the first segment and

$$(s2)_{ik} = K_i - k - 1 + \sum_{r=k+1}^{K_i} c_{ir} \quad (9)$$

is the sum of conditions and fixed blank squares in the second segment.

Similarly, for a column, the equivalent equations are:

$$C_{ij} = \sum_{k=1}^{K_j} \sum_{m=1}^{c_{jk}} \binom{(L1)_{ijkm} - (s1)_{jk} + k - 1}{(L1)_{ijkm} - (s1)_{jk}} \binom{(L2)_{ijkm} - (s2)_{jk} + K_j - k}{(L2)_{ijkm} - (s2)_{jk}} \quad (10)$$

$$(L1)_{ijkm} = i - c_{jk} + m - 2 \quad (11)$$

$$(L2)_{ijkm} = N - i - m \quad (12)$$

$$(s1)_{jk} = k - 2 + \sum_{r=1}^{k-1} c_{jr} \quad (13)$$

$$(s2)_{jk} = K_j - k - 1 + \sum_{r=k+1}^{K_j} c_{jr} \quad (14)$$

Finally, the a-priori probability that the cell (i, j) in the Japanese puzzle is a 1, can be calculated by means of the following expression:

$$p_{ij} = \frac{R_{ij}C_{ij}}{R_{ij} \cdot C_{ij} + (RT_i - R_{ij})(CT_j - C_{ij})} \quad (15)$$

A small example can clarify the concepts used in the calculation above: Figure 2 shows an example of the calculation using one row and one column of a Japanese puzzle. In the depicted row, there are two conditions of size 2, and in the column there are two conditions of size 2 and 1, respectively. The grid length of the row and column is 6 and 5, respectively. The a-priori probability p_{ij} is calculated for the cell situated in the intersection of the row and column considered.

First, we obtain the total number of cell blocks distributions in the row (RT_i) and in the column (CT_j). Note that, though in the example we show all distributions of the row and column, we only need its total number, not the specific distributions (Equations (1) and (3)). In all these distributions, we emphasize the (i, j) cell, in such a way that we can see the number of distributions in the row i with the (i, j) cell fixed to black, i.e., the value of R_{ij} (Equation (5)). Similarly, for the column j , we can see the distributions corresponding to C_{ij} (Equation (10)). Finally, through of these values obtained in the example, and Equation (15) we can obtain the (i, j) a-priori probability which is equal to 0.2.

As a final remark for this subsection, note that a given cell (i, j) such as $p_{ij} = 1$ or $p_{ij} = 0$, can be set to black or white in the preprocessing step of the logic ad-hoc algorithm.

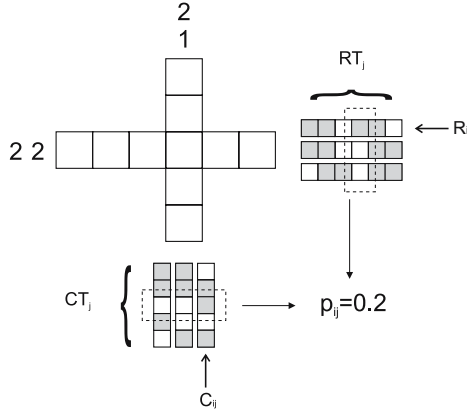


Fig. 2. Example of the a-priori probability calculation in a cell of a Japanese puzzle, given the conditions in the corresponding row and column

3.2 Evolutionary Algorithm Encoding, Initialization and Fitness Function

As was mentioned before, the encoding used in the evolutionary algorithm is a ternary encoding, with values 0 for white cells, 1 for black cells and 2 for unknowns (we encode in the evolutionary algorithm the initialization of the logic ad-hoc algorithm [6]). This logic ad-hoc is then run for each individual in order to obtain the fitness value associated to it. In this case the fitness function is defined as:

$$g(\mathbf{x}) = \sum_{i=1}^N \sum_{j=1}^M (I_{\{x_{ij}=1\}} + I_{\{x_{ij}=0\}}), \quad (16)$$

$$I_{\{x_{ij}=k\}} = \begin{cases} 1 & \text{if } x_{ij} = k \\ 0 & \text{otherwise} \end{cases} \quad (17)$$

which is equivalent to maximize the number of cells fixed to 0 and 1, or minimizing the number of unknown cells. The logic ad-hoc algorithm we use also provides a penalty term in case that the fixed cells by the evolutionary algorithm are incompatible with the puzzle's constraints.

The a-priori probability of bits calculated in the previous subsection can be used to initialize the evolutionary algorithm. We first set a threshold for the number of cells which will be 0 or 1 in the puzzle (the rest will be unknown cells, i.e. fixed to 2). We have set this threshold to be around the 10% of the puzzle's cells. Thus, we first decide for each puzzle's cell, with a probability of 0.1, if that cell is fixed to 0 or 1, or not. If yes, we use the a-priori probability p_{ij} , calculated using Equation (15), to decide if we fix the cell to 1 or 0. In the experiments section we will show that this initialization process improves the convergence properties of the hybrid evolutionary algorithm proposed.

3.3 Evolutionary Operators

Our evolutionary algorithm uses a set of traditional operators: A roulette wheel selection, where the probability of survival of individuals is proportional to its fitness value. We use a two-points crossover, in which, given two individuals, two points are selected on them and the parts of individuals between these two points are swapped from one individual to the other. We also implement a swapping mutation between two cells with different values.

In addition to these traditional operators, we include a novel mutation operator, which uses the a-priori probabilities obtained in Section 3.1, called *Diversity-Guided Mutation* (DGM). The idea of the DGM operator is a mutation in which we take into account both the population diversity and the a-priori probability of each cell in the puzzle. The DGM operator is focussed in each *gen* of the population (a given cell (m, n) in the puzzle), in such a way that only individuals in which this gene is different from 2 are used. We define then a probability for gene i , called q_i , which is the probability that the gene i in an individual is a 1. Note that this q_i is different for each stage of the evolutionary algorithm (depends on the current population) and can be calculated as:

$$q_i = \sum_j \frac{I_{\{g_{ij}=1\}}}{I_{\{g_{ij}=1\}} + I_{\{g_{ij}=0\}}} \quad (18)$$

where g_{ij} stands for the i -th gene in the j -th individual of the population.

$$I_{\{x_j=k\}} = \begin{cases} 1 & \text{if } x_j = k \\ 0 & \text{otherwise} \end{cases} \quad (19)$$

The criterion to decide if we consider that a given gene in the population should be a 1 is the following:

$$q_i \cdot p_{mn} \geq (1 - q_i) \cdot (1 - p_{mn}) \quad (20)$$

which can be easily rewritten with basic operations as:

$$q_i + p_{mn} \geq 1 \quad (21)$$

If this rule is not fulfilled, we consider that the gene should be a 0.

The DGM operator works as follows: If we have decided that a given gene in the population is a 1, we mutate all the genes which are 0s with probability q_i . In the same way, if we have decided that the gene should be a 0, we mutate all the genes set to 1 with probability $(1 - q_i)$. Note that genes set to 2 are not modified by the DGM mutation operator. We will show that this procedure significantly improves the search capacity of the evolutionary algorithm.

4 Experiments and Results

In order to test the performance of the proposed hybrid evolutionary-logic algorithm, we have tackled several Japanese puzzles from [2]. This web site offers

Table 1. Performance of the evolutionary algorithm (EA) proposed in the different Japanese puzzles tackled (in percentage of convergence to the puzzle solution, over 50 runs)

Instance	Proposed EA	EA in [10]	EA no DGM	EA no Init.
222	100	100	100	100
223	100	64	96	98
224	100	54	100	100
225	100	70	98	98
226	100	100	100	100
227	100	100	100	100
228	100	92	94	100
229	100	34	100	100
230	100	14	96	92
231	100	96	96	100
232	98	14	72	92
233	94	0	64	92
237	100	78	96	100
236	96	4	54	94
235	100	0	96	96
234	96	12	86	98

more than 200 Japanese puzzles, sorted by difficulty. A classification of the difficulty with *stars* is used, in such a way that the easiest puzzles are given 1 star, and the most difficult puzzles 9 stars. We have applied our algorithm to tackle all the puzzles of 8 and 9 stars offered in the page (16 puzzles). All these puzzles are 10×10 puzzles, in which the difficulty is in that no initial black nor white cell can be fixed at the beginning of a logic-type algorithm (preprocessing stage).

We compare the performance of our hybrid evolutionary-logic algorithm against the GA in [10] (population of 30 individuals in both algorithms). A comparison in terms of convergence percentage to the optimal puzzles' solution is carried out (over 50 runs of the algorithms). The results of these experiments can be seen in Table 1. It is easy to see how the proposed hybrid evolutionary-logic algorithm obtains the best results, outperforming the GA in [10] in all the puzzles tackled. The percentage of convergence of our algorithm is over 94% in all the puzzles tested, and reaches the 100% in 12 out of 16 puzzles. The results of the algorithm only implementing the DGM operator (without special initialization) and only with the special initialization (without DGM operator) are also displayed in Table 1 (columns 3 and 4 of the table, respectively). Note that the performance of the algorithm in these cases is worse than the complete algorithm, as expected, but better than the GA in [10]. These data show that the DGM operator is a key feature of the algorithm, which improves significantly its performance. The inclusion of the initialization has influence in improving the algorithm's performance, but less than the DGM operator.

5 Conclusions

In this paper we have presented a novel hybrid evolutionary algorithm for solving difficult Japanese puzzles. The evolutionary algorithm encodes the initialization of a logic-type algorithm, and tries to optimize its outcome, in such a way that it is the final solution to the puzzle. Two novel operators, one for initializing the evolutionary algorithm and a second one providing a novel type of mutation adapted to Japanese puzzles have been described in the paper. The proposed evolutionary algorithm has been successfully applied to the solution of several Japanese puzzles instances classified as “very difficult” in a well known web page devoted to these puzzles. A comparison with an existing EA has also been carried out, showing again the good performance of the proposed EA.

References

1. Conceptis Puzzles Inc., <http://www.conceptispuzzles.com>
2. Benchmark Puzzles,
<http://homepages.cwi.nl/~aeb/games/jpuzzlegraded/index.html>
3. Dorant, M.: A beginner's guide to solving picture forming logic puzzles,
<http://www.conceptispuzzles.com/index.aspx?uri=info/article/79>
4. Ueda, N., Nagao, T.: NP-completeness results for nonograms via parsimonious reductions. Internal Report, University of Tokyo, Computer Science Department (1996)
5. Benton, J., Snow, R., Wallach, N.: A combinatorial problem associated with nonograms. *Linear Algebra and its Applications* 412(1), 30–38 (2006)
6. Salcedo-Sanz, S., Ortiz-García, E., PérezBellido, A., Portilla-Figueras, J., Yao, X.: Solving Japanese Puzzles with Heuristics. In: *IEEE Symposium on Computational Intelligence and Games*, Honolulu, USA (April 2007)
7. Ortiz-García, E., Salcedo-Sanz, S., Leiva-Murillo, J.M., PérezBellido, A., Portilla-Figueras, J.: Automated generation and visualization of picture-logic puzzles. *Computers & Graphics* 31, 750–760 (2007)
8. Batenburg, B., Kusters, W.: A discrete tomography approach to Japanese puzzles. In: *Proceedings of the Belgian-Dutch Conference on Artificial Intelligence*, pp. 243–250 (2004)
9. Wiggers, W.: A comparison of a genetic algorithm and a depth first search algorithm applied to Japanese nonograms. In: *Proceedings of the 1st Twente Student Conference on IT*, pp. 1–6 (2004)
10. Salcedo-Sanz, S., Portilla-Figueras, J., PérezBellido, A., Ortiz-García, E., Yao, X.: Teaching advanced features of evolutionary algorithms using Japanese puzzles. *IEEE Transactions on Education* 50(2), 151–155 (2007)
11. Duncan, G.: Puzzle Solving. B.Sc. Degree Final Project Report, University of York, Computer Science Department (1999)
12. Simpson, S.:
<http://www.comp.lancs.ac.uk/computing/users/ss/nonogram/index.html>

Joint Multicast Routing and Channel Assignment in Multiradio Multichannel Wireless Mesh Networks Using Simulated Annealing

Hui Cheng and Shengxiang Yang

Department of Computer Science, University of Leicester
University Road, Leicester LE1 7RH, United Kingdom
{hc118,s.yang}@mcs.le.ac.uk

Abstract. This paper proposes a simulated annealing (SA) algorithm based optimization approach to search a minimum-interference multicast tree which satisfies the end-to-end delay constraint and optimizes the usage of the scarce radio network resource in wireless mesh networks. In the proposed SA multicast algorithm, the path-oriented encoding method is adopted and each candidate solution is represented by a tree data structure (i.e., a set of paths). Since we anticipate the multicast trees on which the minimum-interference channel assignment can be produced, a fitness function that returns the total channel conflict is devised. The techniques for controlling the annealing process are well developed. A simple yet effective channel assignment algorithm is proposed to reduce the channel conflict. Simulation results show that the proposed SA based multicast algorithm can produce the multicast trees which have better performance in terms of both the total channel conflict and the tree cost than that of a well known multicast algorithm in wireless mesh networks.

1 Introduction

Wireless mesh networks (WMNs) [1] have emerged as a new paradigm of static multi-hop wireless networks. Multicast [2] is an important network service, which is the delivery of information from a source to multiple destinations simultaneously. Quality of Service (QoS) requirements [2] proposed by different multicast applications are often versatile. Among them, end-to-end delay [3] is a pretty important QoS metric since real-time delivery of multimedia data is often required. The multicast tree cost, used to evaluate the utilization of network resource, is also an important QoS metric especially in wireless networks where limited radios and channels are available. So far, few work has addressed QoS multicast in WMNs. However, it is believed that efficient multicast, which cannot be readily achieved through combined unicast or simplified broadcast, is essential to WMNs and deserves a thorough investigation [4].

In WMNs, the wireless interference occurs when two links whose distance is less than 2 hops away are assigned to the same channel to support the concurrent communications, which is termed as channel conflict [5]. Therefore, for

multicast routing, each link on the multicast tree requires to be assigned to one channel and the assignment should lead to minimum interference. In fact, the minimum-interference channel assignment problem itself is basically the Max K -cut problem [6], which is known to be NP-hard. Therefore, our problem, i.e., the routing tree construction plus minimum-interference channel assignment, is also NP-hard. In this paper, we propose an efficient QoS multicast routing algorithm in WMNs, which utilizes the powerful simulated annealing (SA) technique to search a low cost routing tree on which the channel assignment can produce the minimum interference. Intuitively by exploring the strong search capability of SA, more candidate routing trees can be examined to help find the one with the minimum channel conflict.

2 Related Work

In [4], two multicast algorithms were proposed, which first build a multicast tree between the source and receivers, and use dedicated strategies to assign channels to the tree aiming to reduce interference. However, since both algorithms separate the construction of the multicast tree and the channel assignment, they will bear a potential drawback, that is, channel assignment cannot work well with the determined multicast tree. Furthermore, they do not consider the delay constraint which is a common issue for multicast problems.

Simulated annealing algorithm simulates the annealing process in the physics of solids. It is observed that a metal body heated to high temperature cools slowly and tends to a state with the least internal energy. SA regards the optimization problem as a physical system and the value of the objective function as its internal energy. With this analogy, annealing is the process of determining a solution with the least value of the objective function [7]. Simulated annealing algorithm is a powerful tool to solve the combinatorial optimization problems. It has been applied to the QoS multicast routing in the wired networks such as the multimedia communication networks [2,8].

We are not aware of any other work that jointly considers multicast routing, which further consists of channel assignment as well as QoS in multiradio multichannel WMNs, although there are quite a few works that are related to some relevant aspects. Since SA has shown good performance in the wired networks, we believe its strong search capability can also help find a low cost low interference routing tree in wireless mesh networks. However, to our best knowledge, SA has not been addressed to solve the multicast problems in WMNs.

3 Problem Formulation

We consider a wireless mesh network with stationary mesh routers where each router is equipped with a certain number of radio network interface cards (NICs). We model a wireless mesh network by a undirected and connected topology graph $G(V, E)$, where V represents the set of mesh routers and E represents the set of communication links connecting two neighboring mesh routers falling into the

radio transmission range. A communication link (i, j) can not be used for packet transmission until both node i and node j have a radio interface each with a common channel. In addition, message transmission on a wireless communication link will experience a remarkable delay.

For clarity of presentation, we assume the *binary interference model*, i.e., two communication links either interfere or do not interfere. Given the binary interference model, the set of pairs of communication links that interfere with each other over the same channel can be represented by a conflict graph [5]. A communication link in the topology graph corresponds to a vertex in the conflict graph. With the binary interference model, the conflict graph $G_c(V_c, E_c)$ can be easily derived from the topology graph $G(V, E)$. We assume the communication links (a, b) and (c, d) in the topology graph $G(V, E)$ are represented by the node i_c and node j_c in the conflict graph $G_c(V_c, E_c)$, respectively. Then if the minimum distance between (a, b) and (c, d) is less than 2 hops, we have $(i_c, j_c) \in E_c$. Here, we summarize some notations that we use throughout this paper.

- $G(V, E)$, the WMN topology graph.
- $G_c(V_c, E_c)$, the conflict graph derived from the WMN topology graph.
- $K = \{0, 1, 2, \dots, k\}$, the set of available orthogonal channels.
- s , the source node of the multicast communication.
- $R = \{r_0, r_1, \dots, r_m\}$, the set of receivers of the multicast communication.
- $T(V_T, E_T)$, a multicast tree with nodes V_T and links E_T .
- V_T^{Leaf} , the set of leaf nodes on the tree T .
- $P_T(s, r_i)$, a path from s to r_i on the tree T .
- d_l , the delay on the communication link l .
- $I_T(f)$, the total channel conflict on the tree T .
- C_T , the cost of the tree T .

The problem of joint QoS multicast routing and channel assignment in a multiradio multichannel wireless mesh network can be informally described as follows. Given a network of mesh routers with multiple radio interfaces, a delay upper bound, a source node and a set of receivers, we wish to find a delay-bounded multicast tree and assign a unique channel to each communication link on the tree. We define the *total channel conflict* as the number of pairs of communication links on the tree that are interfering (i.e., are assigned the same channel and are connected by an edge in the conflict graph). The objective of our problem is to minimize the above defined total channel conflict, as it results in improving the system throughput [4].

We also want to optimize the usage of the scarce network resources in the multicast tree. So we define the *tree cost* as the number of the radio interfaces involved in the multicast communications. We aim to find a multicast tree with low cost. There are two factors related to the tree cost. One is the number of communication links on the tree. Each communication link has one sender and one receiver, thereby occupying two radio interfaces. So we should reduce the number of links on the multicast tree, which also helps reduce the multicast end-to-end delay. The other factor is the number of broadcast nodes generated from the channel assignment. We make all the branch nodes become broadcast

nodes by exploiting wireless multicast advantage (WMA) [9] and the detail is described in Section 4.2. If there are several multicast trees which have the same channel conflict value, we will choose the one with the minimum tree cost.

More informally, consider a wireless mesh network $G(V, E)$ and a multicast communication request from the source node s to a set of receivers R with the delay upper bound Δ . The *joint QoS multicast routing and channel assignment problem* is to find a multicast tree $T(V_T, E_T)$ satisfying the delay constraint as shown in (1) and compute a function $f : E_T \rightarrow K$ defined in (2) to minimize the *total channel conflict* $I_T(f)$ defined in (3).

$$\max_{r_i \in R} \left\{ \sum_{l \in P_T(s, r_i)} d_l \right\} \leq \Delta. \quad (1)$$

$$f(i_c \in E_T) = \{j | j \in K\} \quad (2)$$

$$I_T(f) = |\{(i_c, j_c) \in E_c | f(i_c) = f(j_c), i_c \in E_T, j_c \in E_T\}|. \quad (3)$$

Since the source only transmits packets and all the leaf nodes only receive packets, each of them occupies one radio interface only. All the other nodes are branch nodes which need to do both the transmission and reception. So each branch node occupies two radio interfaces. As a result, the tree cost C_T is calculated as follows:

$$C_T = |\{s\}| + |V_T^{Leaf}| + 2 * (|V_T| - |\{s\}| - |V_T^{Leaf}|). \quad (4)$$

4 Algorithm Design

We adapt SA to the joint multicast routing and channel assignment problem, and the objective function is just the fitness function, which returns the total channel conflict of the multicast tree. The fitness value just simulates the internal energy. First, the initial solution is generated randomly to explore the diversity. Then we start the annealing process at a high temperature. As the temperature decreases, the annealing process tries to converge to the optimal solution. At each temperature, the algorithm searches a number of solutions in the solution space so that the current optimal solution stabilizes at a fitness value. When the temperature decreasing number reaches a specified upper bound and the current optimal solution keeps unchanged, the algorithm terminates and outputs the current optimal solution as the final solution.

4.1 Design of the Simulated Annealing Algorithm

Solution Representation and Initial Solution. A routing path is encoded by a string of positive integers that represent the IDs of nodes through which the path passes. Each locus of the string represents an order of a node. The first

locus is for the source and the last one is for the receiver. The length of a routing path should not exceed the maximum length $|V|$, where V is the set of nodes in the WMN.

For a multicast tree T spanning the source s and the set of receivers R , there are $|R|$ routing paths all originating from s . Therefore, we encode a tree by an integer array in which each row encodes a routing path along the tree. For example, for T spanning s and R , row i in the corresponding array A lists up node IDs on the routing path from s to r_i along T . Therefore, A is an array of $|R|$ rows. All the solutions are encoded under the delay constraint. In case it is violated, the encoding process is usually repeated so as to satisfy the delay constraint.

To explore the solution diversity, in the initial solution Q , all the routing paths are randomly generated. We start to search a random path from s to $r_i \in R$ by randomly selecting a node v_1 from $N(s)$, the neighborhood of s . Then we randomly select a node v_2 from $N(v_1)$. This process is repeated until r_i is reached. Thus, we get a random path $P_T(s, r_i) = \{s, v_1, v_2, \dots, r_i\}$. Since no loop is allowed on the multicast tree, the nodes that are already included in the current tree are excluded, thereby avoiding reentry of the same node. The initial solution is generated as follows.

Step 1: Start($j=0$, $V_T=\emptyset$, $E_T=\emptyset$);

Step 2: Search a random path $P_T(s, r_i)$ which can guarantee $T \cup P_T$ be an acyclic graph;

Step 3: Add all the nodes and links in P_T into V_T and E_T , respectively;

Step 4: $j = j+1$. If $j < |R|$, go to *Step 2*, otherwise, stop.

Fitness Function. Given a solution, we should accurately evaluate its quality (i.e., fitness value), which is determined by the fitness function. In our algorithm, we aim to find a low cost multicast tree on which the minimum interference channel assignment can also be achieved. Our primary criterion of solution quality is the total channel conflict and the subsidiary one is the tree cost. Therefore, among a set of candidate solutions (i.e., multicast trees) with the same minimum channel conflict value, we choose the one with the lowest tree cost. The fitness value of chromosome Ch_i (representing multicast tree T), denoted as $F(Ch_i)$, is given by:

$$F(Ch_i) = [I_T(f) + 1.0]^{-1}. \quad (5)$$

The proposed fitness function only involves the total channel conflict. As mentioned in Section 3, The tree cost is used in the course of selecting the elitism for keeping the searched optimal solution.

Neighborhood Structure. Since SA performs searching from one solution to one of its neighbors in the solution space, we need to determine the neighborhood structure of each solution. In accordance with the solution representation, we propose two methods to construct the neighborhood.

(a) First, randomly select one receiver r_i from R , and randomly select another node v_i on the path ($s \rightarrow r_i$). Then replace the subpath ($v_i \rightarrow r_i$) by a new random subpath.

(b) First, randomly select two receivers r_i and r_j from R , and randomly select another two nodes v_i and v_j on the paths $(s \rightarrow r_i)$ and $(s \rightarrow r_j)$, respectively. Then replace the subpaths $(v_i \rightarrow r_i)$ and $(v_j \rightarrow r_j)$ by new random subpaths, respectively.

Given the current solution, a new neighbor solution will be produced using either of the above two methods. The first method only changes one path on the tree while the second method changes two paths at the same time. Intuitively, the adjustment to the tree is relatively smaller in (a) than in (b). So we name the first method as the fine-grain adjustment and the second method as the coarse-grain adjustment. In the proposed algorithm, we apply the fine-grain adjustment in the first half of the temperature decreasing procedure, and then the coarse-grain adjustment in the second half of the temperature decreasing procedure. Therefore, we can not only guarantee the algorithm converges to the optimal solution theoretically, but also accelerate the procedure to improve the efficiency.

Initial Temperature. We start the SA algorithm from a high temperature (T_0) in order to allow acceptance of any new neighbor solution. A reasonable setting of the initial temperature will reduce the waste of the search time and still allow virtually all proposed uphill or downhill moves to be accepted [8]. In this algorithm, we set $T_0 = 100$.

Temperature Decreasing. We employ the following method:

$$T_{k+1} = \alpha * T_k \quad (0 \leq k, 0 < \alpha < 1) . \quad (6)$$

This method is widely used, simple but effective. By this method, the temperature decreases at the same ratio.

Iterative Length at Each Temperature. In our algorithm, the iterative length at one temperature is proportional to the number of temperature decreasing counted so far. We use L_i to denote the maximum iteration number allowed at temperature T_i , and M_i to denote the maximum number of continuous iterations without improving the present optimal solution allowed at T_i . As the temperature gradually decreases to T_i , both L_i and M_i should become larger simultaneously to explore more candidate solutions in the solution space.

We employ the method of linear increasing, that is, the maximum iteration number allowed at temperature T_i is in direct proportion to the up-to-now times of temperature decreasing, and the maximum number of continuous iterations without improving the present optimal solution allowed at T_i is in direct proportion to the maximum iteration number allowed at the same temperature. The method is formulated as follows:

$$L_i = (i + 1) * \delta * \tau . \quad (7)$$

$$M_i = \omega * L_i . \quad (8)$$

Here, τ is the size of the receiver set, serving as the cardinal number. Since in each iteration, we need to change the path to one receiver. Ideally, we hope the

paths to all the receivers will undergo the change at the same temperature. L_i limits the iteration number at the same temperature to speed up the convergence, and M_i helps stop the iteration at T_i since the search may be stuck in the local optimum.

Termination Rule. The termination rule employed in this algorithm is to control the maximum number of continuous temperature decreasing without improving the present optimal solution. Let the maximum number of temperature decreasing be I , and the upper bound of the continuous temperature decreasing without improving the present optimal solution be U . They have the following relationship:

$$U = \lambda * I \quad (0 < \lambda < 1) . \quad (9)$$

In the proposed algorithm, during the first half period of temperature decreasing, i.e., from T_0 to $T_{\lfloor I/2 \rfloor}$, we generate a neighbor solution by the fine-grain method; during the second half period of temperature decreasing, i.e., from $T_{\lfloor I/2 \rfloor + 1}$ to T_I , we generate a neighbor solution by the coarse-grain method. During the first half period, it is more likely that the difference between the current solution and the global optimal solution is relatively large. So we change two paths to two receivers at each iteration. During the second half period, the difference may become smaller. So we change only one path at each iteration. This design philosophy can help reduce the overhead of the fitness function calculation. Moreover, the algorithm can be theoretically assured to find the global optimal solution as the iteration approaches infinity.

4.2 Channel Assignment Algorithm

In a wireless mesh network, a link cannot be used for data transmission until it has been assigned a wireless communication channel. To support the multicast communication over the routing tree, an appropriate channel should be assigned to each link on the tree so as to achieve the minimum interference (i.e., channel conflict). In addition, the number of available channels is limited in the current network protocols. For example, in IEEE 802.11-based wireless networks, there are 11 available channels. However, at most 3 of them are orthogonal (non-interfering). The number of radio interfaces is also limited as a type of scarce radio network resource. Hence the channel assignment should use as small number of channels and radio interfaces as possible.

Since the minimum-interference channel assignment problem is NP-hard, we propose a heuristic algorithm which aims to reduce both the channel conflict and resource utilization. Given the set of orthogonal channels $K = \{0, 1, \dots, k\}$ ($k \geq 2$), the algorithm works on the multicast tree T as follows.

Step 1) Start($i=0$).

Step 2) Assign channels to the routing path $P_T(s, r_i) = (s, v_1, v_2, \dots, v_{j-1}, r_i)$. Here v_0 represents the source s and v_j represents the receiver r_i , respectively.

- a) Start($n=0$);
 - b) If link (v_n, v_{n+1}) has not been assigned a channel, assign channel $n\%3$ to it;
 - c) $n=n+1$. If $n < j$, go to b).
- Step 3)* $i=i+1$. If $i < |R|$, go to *Step 2*, otherwise, stop.

For each routing path, the algorithm uses 3 channels to do the assignment. Since the minimum distance between two links to avoid channel conflict is 2 hops, 3 is the least number of channels to achieve conflict free assignment on each routing path of the multicast tree. By our assignment strategy, all the links originating from the same branch node are assigned the same channel as utilizes the so-called WMA [9]. WMA refers to that a single transmission can be received by all the nodes that are within the transmission range of a transmitting node. Hence, using one radio interface only, the branch node transmits packets to all its children. This also saves the number of used radio interfaces.

5 Performance Evaluation

In this section, the proposed SA-based joint multicast routing and channel assignment algorithm is compared with Zeng' Level Channel Assignment (LCA) multicast algorithm [4] through simulation experiments. The LCA multicast algorithm is composed of two components. First, it constructs a multicast tree based on breadth first search (BFS) aiming to minimize the hop count distances between the source and receivers. Second, it uses a dedicated strategy to assign channels to the tree aiming to reduce the interference. Hence, this algorithm separates the multicast tree construction and channel assignment. If the channel assignment strategy cannot work well on the generated multicast tree, the algorithm can do nothing while our algorithm can search other trees.

In our algorithm, we have the following SA parameters: the initial temperature (T_0), the coefficient of temperature decreasing (α), the coefficient of the maximum iteration number allowed at one temperature (δ), the coefficient of the maximum number of continuous iterations without improving the present optimal solution allowed at one temperature (ω), and the coefficient of the maximum number of continuous temperature decreasing without improving the present optimal solution (λ). Their suggested values are set in Table 1. In addition, the delay upper bound Δ is set to 20.

Without loss of generality, we assume each mesh router has two radio network interface cards: one for transmission and the other for reception. We assume there are 3 orthogonal channels as the case in 802.11 wireless network. We evaluate both algorithms on two different network topologies: one consists of 11 nodes and 20 links and the other consists of 23 nodes and 34 links. The metrics that we have evaluated include the total channel conflict and the tree cost.

We have compared the SA multicast algorithm with the LCA multicast algorithm over various size of multicast groups. In the WMN of 11 nodes, the size ranges from 3 to 7 whilst in the WMN of 23 nodes it ranges from 3 to 11. Fig. 1

Table 1. SA parameters and the suggested values

Parameter variable	Parameter description	Suggested value
T_0	the initial temperature	100
α	the coefficient of temperature decreasing	0.95
δ	the coefficient of the maximum iteration number allowed at one temperature	1
ω	the coefficient of the maximum number of continuous iterations without improving the present optimal solution allowed at one temperature	0.50
λ	the coefficient of the maximum number of continuous temperature decreasing without improving the present optimal solution	0.30

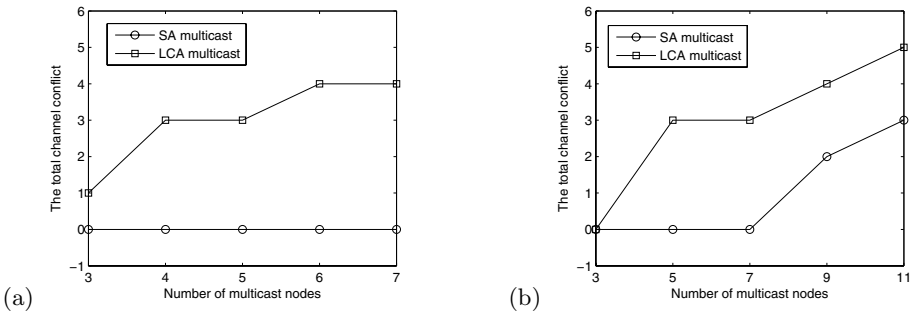


Fig. 1. Comparison of SA multicast and LCA multicast in terms of the total channel conflict in: (a) a WMN of 11 nodes; (b) a WMN of 23 nodes

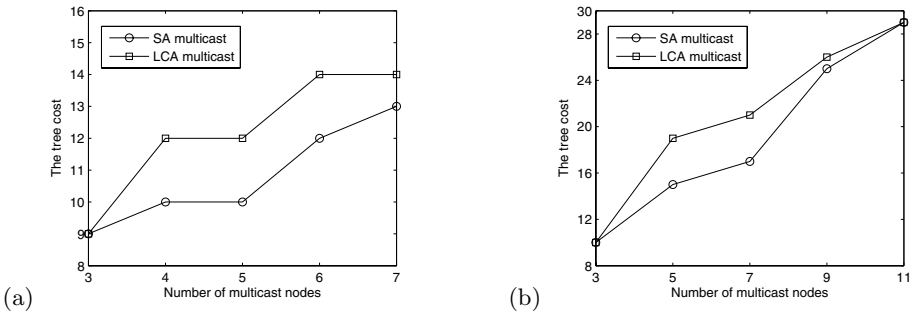


Fig. 2. Comparison of GA multicast and LCA multicast in terms of the tree cost in: (a) a WMN of 11 nodes; (b) a WMN of 23 nodes

and Fig. 2 show the comparison results in terms of the total channel conflict and the tree cost, respectively. From Fig. 1, we can see that in both networks, our SA multicast algorithm can find the multicast trees with less channel conflict than the trees obtained by the LCA multicast algorithm. In the network of 11 nodes, the SA multicast algorithm can always find the conflict-free multicast trees. Furthermore, with the increase of the multicast group size, the LCA multicast algorithm produces the multicast trees with more and more channel conflict. Fig. 2 shows that the cost of our SA multicast trees is also lower than the cost of the LCA multicast trees when the multicast group size exceeds 3. It means that the SA multicast trees occupy less radio network resources. To sum up, our SA multicast algorithm can find the multicast tree which logs less channel conflict and lower cost while satisfying the delay constraint.

6 Conclusions

A routing tree with orthogonal channels appropriately assigned is preferred to support the multicast service in WMNs. However, the optimal multicast routing and channel assignment problem is proved to be NP-hard. This paper presents a simulated annealing algorithm based joint multicast routing and channel assignment algorithm to discover a delay-bounded minimum-interference low cost multicast tree. We believe that the synergy achieved by combining the strong search capability of SA and the effective channel assignment results in the improved quality of solution. We compare the performance of the proposed algorithm with the prestigious LCA multicast algorithm. Experimental results demonstrated that our SA multicast algorithm is capable of finding the multicast trees which have both less channel conflict and lower cost (i.e., occupying less radio network interfaces) than the multicast trees produced by the LCA multicast algorithm.

Acknowledgement

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of UK under Grant EP/E060722/1.

References

1. Akyildiz, I., Wang, X., Wang, W.: Wireless Mesh Networks: a Survey. *Comput. Netw.* 47, 445–487 (2005)
2. Wang, X., Cao, J., Cheng, H., Huang, M.: QoS Multicast Routing for Multimedia Group Communications Using Intelligent Computational Methods. *Comput. Comm.* 29, 2217–2229 (2006)
3. Parsa, M., Zhu, Q., Garcia-Luna-Aceves, J.: An Iterative Algorithm for Delay-constrained Minimum-cost Multicasting. *IEEE/ACM Trans. Netw.* 6, 461–474 (1998)
4. Zeng, G., Wang, B., Ding, Y., Xiao, L., Mutka, M.: Multicast Algorithms for Multi-Channel Wireless Mesh Networks. In: *Proc. ICNP 2007*, pp. 1–10 (2007)

5. Jain, K., Padhye, J., Padmanabhan, V., Qiu, L.: Impact of Interference on Multi-hop Wireless Network Performance. In: Proc. MobiCom 2003, pp. 66–80 (2003)
6. Subramanian, A., Gupta, H., Das, S.: Minimum Interference Channel Assignment in Multi-Radio Wireless Mesh Networks. In: Proc. SECON 2007, pp. 481–490 (2007)
7. Levanova, T.V., Loresh, M.A.: Algorithms of Ant System and Simulated Annealing for the p-median Problem. Autom. Rem. Contr. 65, 431–438 (2004)
8. Zhang, K., Qi, Y., Zhang, H.: Dynamic Multicast Routing Algorithm for Delay and Delay Variation-bounded Steiner Tree Problem. Knowl.-Based Syst. 19, 554–564 (2006)
9. Thai, M., Li, Y., Du, D.: A Combination of Wireless Multicast Advantage and Hitch-hiking. IEEE Commun. Lett. 9, 1037–1039 (2005)

General Game Playing with Ants

Shiven Sharma, Ziad Kobti, and Scott Goodwin

Department of Computer Science, University of Windsor,
401 Sunset Avenue, Windsor, ON N9C4B9, Canada
{sharmaw,kobti,sgoodwin}@uwindsor.ca
<http://www.cs.uwindsor.ca>

Abstract. General Game Playing (GGP) aims at developing game playing agents that are able to play a variety of games and, in the absence of pre-programmed game specific knowledge, become proficient players. The challenge of making such a player has led to various techniques being used to tackle the problem of game specific knowledge absence. Most GGP players have used standard tree-search techniques enhanced by automatic heuristic learning, neuroevolution and UCT (Upper Confidence bounds applied to Trees) search, which is a simulation-based tree search. In this paper, we explore a new approach to GGP. We use an Ant Colony System (ACS) to explore the game space and evolve strategies for game playing. Each ant in the ACS is a player with an assigned role, and forages through the game's state space, searching for promising paths to victory. Preliminary results show this approach to be promising. In order to test the architecture, we create matches between players using the knowledge learnt by the ACS and random players.

Keywords: General Game Playing, Ant Colony System, Adaptive Systems.

1 Introduction

Historically, game playing agents were designed to be good in specific games. Knowledge and heuristics were designed by human players with knowledge and experience of the game and programmed into these agents. However, even though these players excelled in games that they were designed for, they could not play any other games. General Game Playing (GGP) focuses on the creation of agents that are able to accept rules of a game, and use them to learn how to play it, eventually displaying a high level of competence in it. This approach to game playing opens up many areas of challenging research, since the design of a successful game player must take into account aspects such as learning, knowledge representation, reasoning and pattern recognition.

1.1 Positional Games: An Early Attempt at General Game Playing

A class of games for which a GGP approach was taken were *positional games*, which were formalised by [1]. Some examples of positional games include Tic-Tac-Toe, Hex and the Shannon switching games. A positional game can be defined

by three sets, P , A , B . Set P is a set of positions; with set A and B both containing subsets of P . In other words, sets A and B represent a collection of subsets of P , with each subset representing a specific positional situation of the game. The game is played with two players, with each player alternating in moves, which consist of choosing an element from P . The chosen element cannot be chosen again. The aim for the first player is to construct one of the sets belonging to A , whereas the aim for the second player is to construct one of the sets belonging to B . Programs that are capable of accepting rules of positional games and, with practice, learn how to play the game have been developed. [1] constructed a program that is able to learn important board configurations in a 4 x 4 x 4 Tic-Tac-Toe game. This program plays about 12 times before it learns and is effectively able to play and start defeating opponents. A set of board configurations are described by means of a weighted graph.

1.2 Current Approaches to General Game Playing

The annual General Game Playing Competition [2] organised by Stanford University has been instrumental in bringing about renewed interest in GGP. The rules of the games are written in Game Description Language (GDL) [3], which is syntactically similar to prefix KIF [4]. The tournaments are controlled by the Game Manager (GM) which relays the game information to each Game Player (GP) and checks for legality of moves and termination of the game. Communication between players and the GM takes place in the form of HTTP messages. A more detailed description of the architecture and game rules can be found at [5]. Successful players have mainly focused on automatically generating heuristics based on certain generic features identified in the game. Cluneplayer [6] was the winner of the first GGP competition, followed by Fluxplayer [7]. Both these players, along with UTexas Larg [8] use automatic feature extraction. Evaluation functions are created as a combination of these features and are updated in real-time to adapt to the game. Another approach that has been taken is in [9], where transfer of knowledge extracted from one game to another is explored by means of a $TD(\lambda)$ based reinforcement learner. CADIA-Player [10] was the first General Game Player to use a simulation based approach, using UCT [11] to search for solutions, and was the winner of the last GGP Competition.

The main aim of this paper is to explore a new approach to GGP. Traditionally, Ant Colony Systems [12] have been used in solving optimisation problem, such as the Travelling Salesman Problem [13]. [14] has developed an ACS that plays Chess, which demonstrated the emergence of strategies. [15] discusses the use of Ants in Computer Role Playing Games (CRPG). In this paper, we explore the use of ACS for creating a General Game Player. Each agent (an Ant) is assigned a unique role of the game, and starts playing the game randomly. Paths are composed of a state and a possible action (move) that can be taken from it. As generations progress, successful paths are reinforced by pheromone deposits and desirability criteria. In this manner, the ants explore the game space and mark out strong paths. A game playing agent uses these paths to play the game. Preliminary results show this approach to be promising.

The rest of the paper is organised as follows. In Section 2 we give a brief introduction to Ant Colony Systems (ACS). In Section 3 we discuss our approach to GGP using ACS. Matches are then played between a player employing the knowledge generated by the ants and a random player for small (3 x 3) and large (5 x 5) Tic-Tac-Toe, Connect-4, Breakthrough and Checkers. Finally, we discuss directions for future work.

2 Ant Colony Systems

Ant Colony Systems (ACS) were developed by [12]. They are inspired from the behaviour of ants in nature. In nature, ants wander randomly, searching for food. Once they have found food, they return to their colony while laying down pheromone trails. These act as guides for other ants in the future. When other ants find such a path, instead of wandering around randomly, they are more likely to follow the trail and further reinforce it by their pheromone deposits if they are successful. Since pheromone evaporates over time, shorter paths are more likely to have a stronger concentration of deposits. As a consequence, over time, short paths get favoured by more and more ants. This approach is applied in computer science to solve optimisation and path finding problems, using multiple agents (the ants) that move around in the problem space in search for the desired solutions.

Two key parameters that determine the state transitions are the *desirability* (or attractiveness), η_{ij} , and *pheromone level*, τ_{ij} of the path (or arc) between the two states i and j . η_{ij} is usually represented by a predefined heuristic, and therefore indicates an *a priori* fitness of the path. On the other hand, τ_{ij} indicated the past success of the move, and therefore represents a *posteriori* fitness of the path. The update for τ_{ij} take place once all the ants have finished foraging. Given these two parameters, the probability of selecting a path p_{ij} between states i and j is given by (1)

$$p_{ij} = \frac{(\tau_{ij}^\alpha)(\eta_{ij}^\beta)}{\sum_{\kappa \in M} (\tau_{i\kappa}^\alpha)(\eta_{i\kappa}^\beta)} \quad (1)$$

α and β are user-defined parameters that determine how much influence should be given to the trail strength and desirability respectively. M is the set of all legal moves that can be made from state i .

Once all the ants have finished foraging through the state space, the trails are updated as

$$\tau_{ij}(t) = \rho\tau_{ij}(t-1) + \Delta\tau_{ij} \quad (2)$$

$\Delta\tau_{ij}$ is the cumulative accumulation of pheromone by each ant that has passed between i and j . ρ is called the *evaporation* parameter, and determines by what value the previous trail level decreases. Algorithm 1 gives the basic flow of how the Ant Colony System works.

Apart from the standard ACS described here, many variations of Ant Colony Systems exist. [16] combines ACS with Q-Learning [17]. In Elitist Ant Systems

Algorithm 1. Ant Colony System

```

1: Initialise  $\tau$  and  $\eta$  parameters
2: while Terminal condition is not met do
3:   for all Ant  $a$  in AntColony do
4:     while solution is not complete do
5:       select a transition  $t$  probabilistically
6:     end while
7:   end for
8:   for all Trails  $t_{ij}$  made by all Ants do
9:     update  $\tau_{ij}(t) \leftarrow \rho\tau_{ij}(t-1) + \Delta\tau_{ij}$ 
10:   end for
11: end while

```

[18], the best solution seen globally also deposits pheromone on the trails along with the other ants. Rank-Based Ant Systems (ASrank) [19] have solutions sorted by fitness, and consequently pheromone is weighted for each solution, with stronger solutions depositing more pheromone than weaker solutions. For a comprehensive resource on the subject, see [20].

In the next section, we describe our architecture for the ACS for General Game Playing.

3 Ant Colony System for General Game Playing

Each ant in the Ant Colony System (ACS) is a player that is assigned a unique role as per the rules of the game (an example of a role would be *black* or *white* in Chess). The ant colony is controlled by a central agent which is responsible for creating the ants, assigning them roles, passing them the game rules and storing and updating the pheromone level and desirability of the paths foraged by the ants. The game rules are taken from [5] and are written in GDL (Game Description Language) [3]. The path in this case is simply the sequence of states and the moves the ant has made from these states. Each state-move combination has an associated pheromone deposit and desirability value associated with it. Moves are selected from a state probabilistically based on these values. Fig. 1 shows the basic colony structure. Note that path subsections are basically stored as sets of moves associated with a state from which these moves were made. In case of games with large state-spaces, this becomes impractical, and methods to overcome this are discussed in Section 5.

Unlike the conventional ACS, the desirability of each path is also modified. This is done as there is no pre-defined knowledge available for the game. Therefore, the desirability evolves over time as more and more ants forage through these paths. The ant has had a successful forage if it is able to play moves that ultimately lead it to a winning terminal state. The pheromone and desirability values are updated according to the final outcome of the forage, i.e. the game.

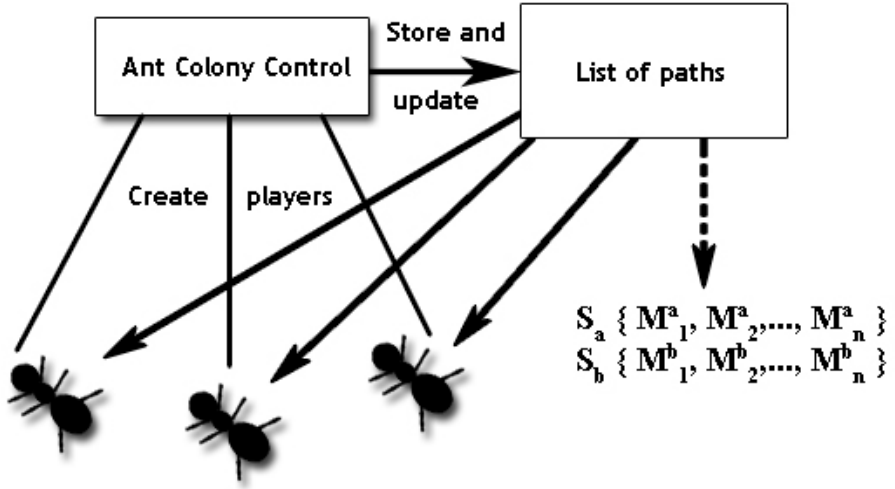


Fig. 1. The basic structure of the Ant Colony for GGP. The Control stores all the paths seen by the ants. The ants use these paths to forage through the game space. Path sections are stored as states with an associated set of moves.

3.1 Description of Ant Colony System

Algorithm 2 gives the outline of the Ant Colony GGP System. As denoted in line 1, all ants are created with a unique role. Consequently, the Ant Colony Control (ACC) maintains a list of paths for each role. If, during game play, it is not the turn of the role assigned to the ant playing the game (as seen in lines 8 through 12), the ant has the option of either making a random move for the opponent role or consulting the corresponding opponent roles' path repository in the ACC. This choice is made probabilistically using a user-defined probability.

Updates to the pheromone trails and desirability for each path take place as follows. We use the average of the final outcomes of the plays (forages) of the game as a value for the pheromone deposit for the state and the corresponding move. The pheromone update takes place as

$$\tau_{sm}(t) = \rho\tau_{sm}(t-1) + \sum_{\gamma \in I} \frac{A_{\gamma}}{|I|} \quad (3)$$

A_{γ} is the final outcome of game sequence γ . I is the set of all game sequences which include move m made from state s .

The desirability of a path is set to reflect the cumulative average score though that path and the average distance, L_{avg} , from that path to a terminal state(s). A single unit of distance is a single turn in the game. Therefore, the desirability is updated as

$$\eta_{sm}(t) = \eta_{sm}(t-1) + \frac{wins}{visits} + \frac{\vartheta}{L_{avg}} \quad (4)$$

Algorithm 2. Ant Colony GGP System

```

1: Initialise each  $Ant \in Ants$  with a unique role
2:  $allGameSequences \leftarrow$  empty list
3: while  $numberOfForages \leq totalForages$  do
4:   for all  $Ant \in Ants$  do
5:      $currentState \leftarrow$  current state of the game
6:      $gameSequence \leftarrow$  empty list
7:     while terminal state of game is not reached do
8:       if not turn of  $Ant.role$  then
9:         make random move  $m$  or consult ACC for move  $m$ 
10:      else if it is turn of  $Ant.role$  then
11:        select move  $m \in legalMoveList$  using (1)
12:      end if
13:       $gameSequence.add(currentState, m)$ 
14:       $currentState \leftarrow updateState(currentState, m)$ 
15:    end while
16:     $allGameSequences.add(gameSequence)$ 
17:  end for
18:  for all Trails  $t_{sm}$  (state  $s$  and move  $m$ )  $\in allGameSequences$  do
19:     $updatePheromone(outcome)$ 
20:     $updateDesire(outcome, distanceFromTerminal)$ 
21:  end for
22:   $allGameSequences \leftarrow$  empty list
23: end while

```

ϑ is a user-defined constant that affect how the distance parameter influences the desirability. Note that while the pheromone is the average outcome for a single set of forages, the desirability includes the average of all outcomes seen during training.

The result of all these ants foraging around is that a table of $\langle state, moveSet \rangle$ values are built, each with its corresponding pheromone and desirability value. This can be visualised as a landscape (which is a subset of the entire game space) of paths of varying thickness (corresponding to the strength). Each of these paths can be considered as a strategy. A player tries to walk on the paths that have the greatest thickness. In other words, the strategies can be used by a player to select which move to make from a given state. Selection can be either greedy or probabilistic. In the next section we describe the experiments that were performed using these strategies.

Fig. 2 gives an illustration of a forage of an ant with the role of marking X, Ant_X in Tic-Tac-Toe. Pheromone is deposited after a series of forages has been completed. Whenever Ant_X is in a state from which the opponent ant has to make a move (ant with role of marking O, Ant_O), it can either ask Ant_O for a move (it uses the knowledge of Ant_O) or simply make a random move.

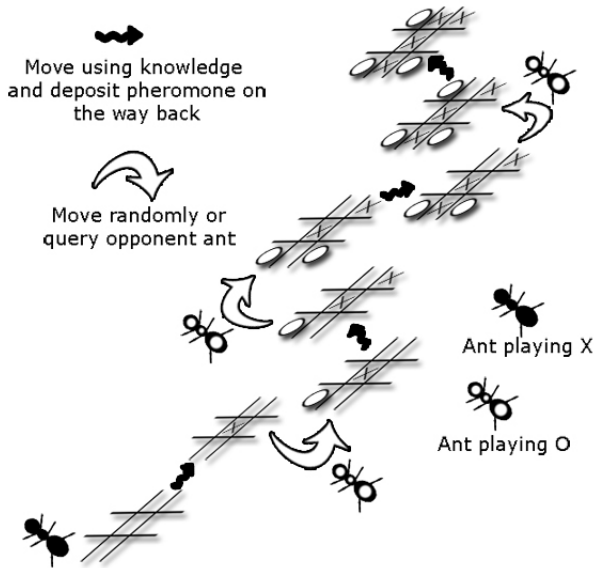


Fig. 2. The forage of Ant_X through the Tic-Tac-Toe landscape. The presence of Ant_O besides the hollow arrows indicates that Ant_X has the option of asking Ant_O for a move, though it is not necessary to do so. Pheromone is deposited along the squiggly arrow once a series of forages has been completed.

4 Experiments

In order to test the basic effectiveness of the Ant Colony architecture, we played matches between a player using the knowledge evolved by the ants, ANT , and a random player, $RAND$, which makes moves randomly. Selections were made greedily with respect to the product of the pheromone and desirability.

A total of 40 ants were created for each game, with the roles equally distributed amongst the ants. The total number of foraging runs was 200. The parameter values that gave the best results are as follows: the value for α was set to 0.6, the value for β to 0.8 and the probability to consult the opponent paths (as given in lines 8 through 12 in Algorithm 2) was set to 0.5. ϑ was set to 1.

In order to have a more controlled environment for testing, we implemented a GGP type system on our local machines. This system is similar to the one implemented at Stanford, and used the same communication protocol between the Game Manager and Game Players. The rules were downloaded from the Stanford website and were passed by the Game Manager to the players. The players and the manager were written in Java. For inferencing, the game rules were converted to Prolog. YProlog [21], a Prolog-style inference engine written in Java, was used. The games used for testing were standard Tic-Tac-Toe, 5 X 5 Tic-Tac-Toe, Connect-4, Breakthrough and Checkers. The results for 100 games each are summarised below. The knowledge collected by the ants was not evolved

Table 1. Number of wins for *ANT* and *RAND* over 100 matches per game

Game	Wins for <i>ANT</i>	Wins for <i>RAND</i>	Total Draws
Tic-Tac-Toe (small)	72	9	19
Tic-Tac-Toe (large)	66	12	22
Connect-4	73	27	0
Breakthrough	74	26	0
Checkers	59	39	2

during game play. The roles for the games were evenly distributed between the players.

The results show that the ants were able to successfully generate strategies that enabled *ANT* to win a majority of games. It is important to note that for large games (such as Checkers), it is not practical to store the knowledge as a table of state-move lists. Function approximation techniques for representing values are more suitable, though at the cost of accuracy. These are discussed briefly in the next section.

Generating and training knowledge for games that have long sequences is relatively time consuming. In an actual competition, it might not be possible to generate a fair knowledge representation if a sufficient amount of time is not given. However, the aim of these experiments was to test the effectiveness of the ACS approach, without taking into account a competitive scenario. The time can still be reduced by using efficient data structures and hashing techniques such as Zobrist hashing [22], as used by [10], for state lookup.

5 Conclusions and Future Work

In this paper we explored the use of Ant Colonies for evolving knowledge and strategies for General Game Players. Preliminary experiments against a random player shows that the colonies are able to evolve and discover promising paths in the game space, thereby enabling players to make intelligent moves. However, using Ant Colonies to play games is not sufficient on its own. The main player we are developing employs the UCT (Upper Confidence Bounds applied to Trees)[11] algorithm. However, the UCT algorithm itself depends heavily on random simulations and selecting nodes appropriately to descend down the tree. In Mo-Go [23], one of the best computer Go players today, domain dependent knowledge is used to improve the performance of UCT. [10] uses a basic move history heuristic [24] to supplement UCT. Therefore, integrating the Ant Colony approach in conjunction with other knowledge generation techniques to supplement the UCT algorithm is an interesting direction for future work.

Using features to approximate value functions of states [25] is useful when the state-space becomes too large. We have also developed a knowledge scheme that uses a modified move history heuristic and also assigns values to states based on features in the states [26]. Also, work done in [27] involved testing a multi-agent architecture to evolve basic sequence patterns by using Monte-Carlo simulations.

With the Ant Colony approach showing promising results in preliminary tests, we plan on combining it with this knowledge scheme and architecture and use the combination with UCT. Future work will involve exploring ways of implementing these combinations.

Acknowledgments. This work was funded in part by an NSERC Discovery grant.

References

1. Koffman, E.: Learning through pattern recognition applied to a class of games.: IEEE Trans. on Systems, Man and Cybernetics SSC-4 (1968)
2. Genesereth, M., Love, N.: General game playing: Overview of the aaai competition. AI Magazine (Spring 2005)
3. Genesereth, M., Love, N.: General game playing: Game description language specification, <http://games.stanford.edu>
4. Genesereth, M.R., Fikes, R.E.: Knowledge interchange format, version 3.0 reference manual. Technical report logic-92-1, Stanford University
5. <http://games.stanford.edu>
6. Clune, J.: Heuristic evaluation functions for general game playing. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (2007)
7. Schiffl, S., Thielscher, M.: Fluxplayer: A successful general game player. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, pp. 1191–1196 (2007)
8. Banerjee, B., Kuhlmann, G., Stone, P.: Value function transfer for general game playing. In: ICML Workshop on Structural Knowledge Transfer for ML (2006)
9. Banerjee, B., Stone, P.: General game playing using knowledge transfer. In: The 20th International Joint Conference on Artificial Intelligence, pp. 672–777 (2007)
10. Björnsson, Y., Finnsson, H.: Simulation-based approach to general game playing. In: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence. AAAI Press, Menlo Park (2008)
11. Kocsis, L., Szepesvari, C.: Bandit based monte-carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS, vol. 4212, pp. 282–293. Springer, Heidelberg (2006)
12. Dorigo, M.: Optimization, Learning and Natural Algorithms. PhD thesis, Politecnico di Milano, Italy (1992)
13. Dorigo, M., Gambardella, L.M.: Ant colony system: A cooperative learning approach to the travelling salesman problem. IEEE Transactions on Evolutionary Computation, 53–66 (1997)
14. Drogoul, A.: When ants play chess (or can strategies emerge from tactical behaviours?). In: Müller, J.P., Castelfranchi, C. (eds.) MAAMAW 1993. LNCS, vol. 957, pp. 11–27. Springer, Heidelberg (1995)
15. Dunn, J.: Ant colony organization for mmorpg and rts creature resource gathering. AI Game Programming Wisdom 3, 495–506 (2006)
16. Dorigo, M., Gambardella, L.M.: Ant-q: A reinforcement learning approach to the travelling salesman problem. In: International Conference on Machine Learning (1995)
17. Watkins, C.: Learning from Delayed Rewards. PhD thesis, Cambridge University, Cambridge, England (1989)

18. White, T., Kaegi, S., Oda, T.: Revisiting elitism in ant colony optimization. In: Genetic and Evolutionary Computation Conference (2003)
19. Zhang, T., Yu, C., Zhang, Y., Tian, W.: Ant colony system based on the asrank and mmas for the vrpspd. In: International Conference on Wireless Communications, Networking and Mobile Computing, pp. 3728–3731 (2007)
20. <http://www.aco-metaheuristic.org/>
21. Winikoff, M.: <http://www3.cs.utwente.nl/~schooten/yprolog>
22. Zobrist, A.: A new hashing method with application for game playing. Technical report 99, University of Wisconsin (1970)
23. Gelly, S., Wang, Y.: Modifications of uct and sequence-like simulations for monte-carlo go. In: IEEE Symposium on Computational Intelligence and Games, Honolulu, Hawaii (2007)
24. Schaeffer, J.: The history heuristic and alpha-beta search enhancements in practice. IEEE Transaction on Pattern Analysis and Machine Intelligence, 1203–1212 (1989)
25. Sutton, R., Barto, A.: Reinforcement Learning, An Introduction. MIT Press, Cambridge (1998)
26. Sharma, S., Kobti, Z., Goodwin, S.: Knowledge generation for improving simulations in uct for general game playing. In: 21st Australasian Joint Conference on Artificial Intelligence (in press, 2008)
27. Sharma, S., Kobti, Z.: A multi-agent architecture for general game playing. In: IEEE Symposium on Computational Intelligence and Games, Honolulu, Hawaii (2007)

A Generalized Approach to Construct Benchmark Problems for Dynamic Optimization

Changhe Li and Shengxiang Yang

Department of Computer Science, University of Leicester
University Road, Leicester LE1 7RH, UK
{c1160,s.yang}@mcs.le.ac.uk

Abstract. There has been a growing interest in studying evolutionary algorithms in dynamic environments in recent years due to its importance in real applications. However, different dynamic test problems have been used to test and compare the performance of algorithms. This paper proposes a generalized dynamic benchmark generator (GDBG) that can be instantiated into the binary space, real space and combinatorial space. This generator can present a set of different properties to test algorithms by tuning some control parameters. Some experiments are carried out on the real space to study the performance of the generator.

1 Introduction

In recent years, there has been a growing interest in studying evolutionary algorithms for dynamic optimization problems (DOPs) due to its importance in real world applications. In order to study the performance of EAs in dynamic environments, one important task is to develop proper dynamic benchmark problems. Over the years, researchers have applied a number of dynamic test problems to compare the performance of EAs in dynamic environments. Generally speaking, they can be roughly divided into two types.

For the first type, the environment is just switched between several stationary problems or several states of a problem. For example, many researchers tested their approaches on the dynamic knapsack problem where the weight capacity of the knapsack changes over time, usually oscillating between two or more fixed values [1].

The second type of DOP generators construct dynamic environments by reshaping a predefined fitness landscape. For example, Branke [2] suggested a dynamic benchmark problem, called the “moving peaks” benchmark (MPB) problem. It consists of a multi-dimensional landscape with several peaks, where the height, width and position of each peak is altered a little every time the environment changes. This function is capable of generating a given number of peaks in a given number of dimensions that vary both spatially (position and shape of a peak) and in terms of fitness. Morrison and De Jong [3] also defined a similar dynamic generator as the MPB problem. Yang and Yao [4,5,6] proposed a DOP generator that can generate dynamic environments from any binary encoded stationary problem using a bitwise exclusive-or (XOR) operator.

Though a number of DOP generators exist in the literature, there is no a unified approach of constructing dynamic problems across the binary space, real space, and combinatorial space so far. This paper proposes a generalized dynamic benchmark generator (GDBG) to construct dynamic environments for all the three solution spaces. GDBG provides six properties of the environmental dynamics, which are random change, small step change, large step change, chaotic change, recurrent change, and recurrent change with noisy. Especially, in the real space, we introduce a rotation method instead of shifting the positions of peaks as in [3] and [2]. The rotation method can overcome the problem of unequal challenge per change for algorithms of the MPB generator, which happens when the peak positions bounce back from the boundary of the landscape. In this paper, some experiments are carried out on the real space using the particle swarm optimization (PSO) [7,8] algorithm and fast evolutionary programming (FEP) [9] to test the performance of the GDBG system.

The rest of the paper is organized as follows. Section 2 describes the GDBG system in details. Section 3 presents several instances of the GDBG in the binary space, the real space and the combinatorial space. Then some experiments based on the real space are carried out to test the performance of the GDBG system in Section 4. Finally, Section 5 concludes the paper.

2 The Generalized Dynamic Benchmark Generator

In this section, we first define DOPs and then introduces the GDBG system. DOPs can be defined as follows:

$$F = f(x, \phi, t) \quad (1)$$

where F is the optimization problem, f is the cost function, x is a feasible solution in the solution set $\mathbf{X}$, t is the real-world time, and ϕ is the system control parameter, which determines the solution distribution in the fitness landscape. The objective is to find a global optimal solution x^* such that $f(x^*) \leq f(x) \forall x \in \mathbf{X}$ (without loss of generality, minimization problems are considered in the paper).

First, we classify the environmental changes into two categories: the dimensional changes and the non-dimensional changes. Dimensional changes correspond to adding or removing variables from the optimization problem. For example, the number of cities increases or decreases in the traveling salesman problem (TSP), the dimensions in the function optimization problem and the number of objects in the knapsack problems increases or decreases. This kind of changes requires the alternation of the representation of solutions.

Non-dimensional changes result from the change of the value of variables within the problem constraints. For example, the capacity of the knapsack changes and the weight or profit of objects changes in the knapsack problem, the positions of cities change in the TSP, the position of peaks change in the function optimization problem, and the processing time and ready date change in a scheduling problem. Non-dimensional changes might be regarded harder than dimensional changes when there's no relationship among dimensions. We just

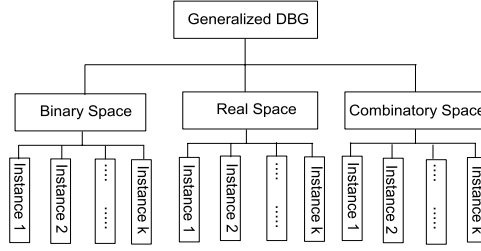


Fig. 1. Model of the GDBG system

simply delete or add variables when dimensional changes occur. In this paper, we only consider non-dimensional changes in the GDBG system.

In the GDBG system, the dynamism results from a deviation of solution distribution from the current environment by tuning the system control parameters. It can be described as follows:

$$\phi(t+1) = \phi(t) \oplus \Delta\phi \quad (2)$$

where $\Delta\phi$ is a deviation from the current system control parameters. Then, we can get the new environment at the next moment $t+1$ as follows:

$$f(x, \phi, t+1) = f(x, \phi(t) \oplus \Delta\phi, t) \quad (3)$$

The system control parameters decide the distribution of solutions in the solution space. They may be different from one specific instance to another instance. For example, the only distance matrix determines the solution distribution in TSP, while the capacity of the knapsack and the weights and profits of objects together determine the solution distribution in the knapsack problem. The GDBG system constructs dynamic environments by changing the values of these system control parameters. Fig. 1 shows the model of the GDBG system.

There are six change types of the system control parameters in the GDBG system. They are small step change, large step change, random change, chaotic change, recurrent change, and recurrent change with noisy. By controlling the values of the system control parameters using the six change types, GDBG can present six different dynamic properties. This easily enables the test and comparison of the adaptability of algorithms in different dynamistic types. The framework of the six change types are described as follows:

Framework of DynamicChanges

switch(change type)

case small step: $\Delta\phi = \alpha \cdot \|\phi\| \cdot r \cdot \phi_{severity}$

case large step: $\Delta\phi = \|\phi\| \cdot (\alpha \cdot \text{sign}(r) + (\alpha_{max} - \alpha) \cdot r) \cdot \phi_{severity}$

case random: $\Delta\phi = N(0, 1) \cdot \phi_{severity}$

case chaotic: $\phi(t+1) = A \cdot \phi(t) \cdot (1 - \phi(t)/\|\phi\|)$

case recurrent: $\phi(t+1) = \phi_{min} + \|\phi\|(\sin(\frac{2\pi}{P}t + \varphi) + 1)/2$

case recurrent with noisy:

$\phi(t+1) = \phi_{min} + \|\phi\|(\sin(\frac{2\pi}{P}t + \varphi) + 1)/2 + N(0, 1) \cdot \text{noisy}_{severity}$

where $\|\phi\|$ is the change range of ϕ , $\phi_{severity} \in (0, 1)$ is change severity of ϕ , ϕ_{min} is the minimum value of ϕ , $noisy_{severity} \in (0, 1)$ is noisy severity in recurrent with noisy change. $\alpha \in (0, 1)$ and $\alpha_{max} \in (0, 1)$ are constant values, which are set to 0.02 and 0.1 in the GDBG system. A logistics function is used in the chaotic change type, where A is a positive constant between (1.0, 4.0), if ϕ is a vector, the initial values of the items in ϕ should be different within $\|\phi\|$ in chaotic change. P is the period of recurrent change and recurrent change with noise, φ is the initial phase, r is a random number in $(-1, 1)$, $sign(x)$ returns 1 when x is greater than 0, returns -1 when x is less than 0, otherwise, returns 0. $N(0, 1)$ denotes a normally distributed one dimensional random number with mean zero and standard deviation one.

In the following section, some specific instances from the GDBG system in the binary space, real space, and combinatorial space are described respectively.

3 Generator Instances from the GDBG system

3.1 Generator Instance in the Binary Space

The XOR DOP generator proposed in [4,5,6] can generate DOPs from any binary encoded stationary problem. Given a stationary problem $f(\mathbf{x}, \phi) (\mathbf{x} \in \{0, 1\}^l)$, where l is the length of the binary representation, $\phi \in [0, l]$ is the number of ones in x . In the GDBG system, we can also use the XOR operator to construct a new environment from the current fitness landscape as follows:

- Step 1. $\phi(t+1) = \text{DynamicChanges}(\phi(t))$
- Step 2. Generate a binary string $\mathbf{m}(t)$ of length l containing $\phi(t+1)$ ones
- Step 3. $\mathbf{x}(t+1) = \mathbf{x}(t) \oplus \mathbf{m}(t)$

where “ $\oplus$ ” is a XOR operator, i.e., $1 \oplus 1 = 0, 1 \oplus 0 = 1, 0 \oplus 0 = 0$. $\phi_{severity} = 1$, $\|\phi\| = l$. The difference between $\phi(t+1)$ and $\phi(t)$ controls the severity of each environmental change. The $\phi(t)$ is predefined in [4,5,6], it is one case from GDBG system when α is set 0.

3.2 Generator Instances in the Real Space

In this section, two different real DBGs will be constructed using different methods. In [2] and [3], two real DBGs were constructed that contain several peaks. The height, width and position of each peak may change every time an environmental change occurs. They have been tested by many researchers. However, both have a disadvantages of unequal challenge per change for algorithms when the position of a peak bounces back from the search boundary. This paper proposes a rotation method for the peak position to overcome that shortcoming.

Real Rotation DBG

The fitness landscape of the rotation DBG also consists of several peaks that can be artificially controlled. The height, width, and position of each peak are

system control parameters, which are altered according to the above six change types. Given a problem $f(x, \phi, t)$, $\phi = (\mathbf{H}, \mathbf{W}, \mathbf{X})$, where $\mathbf{H}$, $\mathbf{W}$ and $\mathbf{X}$ denote the peak height, width and position respectively. The function of $f(x, \phi, t)$ is defined as follows:

$$f(x, \phi, t) = \text{Min}_{i=1}^m (\mathbf{H}_i(t) + \mathbf{W}_i(t) \cdot (\exp(\sqrt{\sum_{j=1}^n \frac{(x_j - \mathbf{X}_j^i(t))^2}{n}}) - 1)) \quad (4)$$

where m is the number of peaks, n is the number of dimensions. $\mathbf{H}$ and $\mathbf{W}$ change as follows:

$$\begin{aligned} \mathbf{H}(t+1) &= \text{DynamicChanges}(\mathbf{H}(t)) \\ \mathbf{W}(t+1) &= \text{DynamicChanges}(\mathbf{W}(t)) \end{aligned}$$

where in the height change, *height_severity* should read $\phi\_h\_severity$ and $\|\phi\_h\|$ is height range. Accordingly, *width_severity* and width range should read $\phi\_w\_severity$ and $\|\phi\_w\|$ in the width change

Instead of shifting peak position as in MPB[2], we borrow the idea from [10] and use rotation matrices to change the peak position in the GDBG system.

A rotation matrix $R_{ij}(\theta)$ is obtained by rotating the projection of $\vec{x}$ in the plane $i - j$ by an angle θ from the i -th axis to the j -th axis. The peak position $\mathbf{X}$ is changed by the following algorithm:

Step 1. Randomly select l dimensions (l is an even number) from the n dimensions to compose a vector $r = [r_1, r_2, \dots, r_l]$.

Step 2. For each pair of dimension $r[i]$ and dimension $r[i+1]$, construct a rotation matrix $R_{r[i], r[i+1]}(\theta(t))$, $\theta(t) = \text{DynamicChanges}(\theta(t-1))$.

Step 3. A transformation matrix $A(t)$ is obtained by:

$$A(t) = R_{r[1], r[2]}(\theta(t)) \cdot R_{r[3], r[4]}(\theta(t)) \cdots R_{r[l-1], r[l]}(\theta(t)), \theta(t) \in (0, 2\pi)$$

Step 4. $\mathbf{X}(t+1) = \mathbf{X}(t) \cdot A(t)$

where the change severity of θ ($\phi\_theta\_severity$) is set 1, the range of θ should read $\|\phi\_theta\|$, $\|\phi\_theta\| \in (0, 2\pi)$

By changing the height, width and position of each peak, we can easily construct a new fitness landscape with different properties. However, this artificial fitness landscape has a problem that the shape of each peak in the fitness landscape is symmetrical at their peak position. It is easy to obtain the local optima for some algorithms using this property. For example, PSO is effective to search the global optimum in symmetrical fitness landscapes. We can see this from the experimental results presented in the paper. So, a composition real DBG based on several static benchmark problems is proposed in this paper.

Real composition DBG

A composition function construction method was proposed in [11]. The idea is to compose the standard benchmark functions to construct a more challenging function with a randomly located global optimum and several randomly located

local optima. By shifting, rotating and composing the global optimum of standard functions, we can get more challenging test functions possessing many desirable properties. In the GDBG system, we get dynamism by controlling the values and locations of these global and local optima. The composition function can be described as:

$$F(x, \phi, t) = \sum_{i=1}^m (w_i \cdot (f'_i((x - \mathbf{O}_i(t) + \mathbf{O}_{iold})/\lambda_i \cdot \mathbf{M}_i) + \mathbf{H}_i(t))) \quad (5)$$

where the system control parameter $\phi = (\mathbf{O}, \mathbf{M}, \mathbf{H})$, $F(x)$ is the composition function, $f_i(x)$ is i -th basic function used to construct the composition function. m is the number of basic functions, $\mathbf{M}_i$ is orthogonal rotation matrix for each $f_i(x)$, $\mathbf{O}_i$ and $\mathbf{O}_{iold}$ are the shifted and old optimum position for each $f_i(x)$. The weight value w_i for each $f_i(x)$ is calculated as:

$$w_i = \exp(-\text{sqrt}(\frac{\sum_{k=1}^n (x_k - o_i^k + o_{iold}^k)^2}{2n\sigma_i^2}))$$

$$w_i = \begin{cases} w_i & \text{if } w_i = \max(w_i) \\ w_i \cdot (1 - \max(w_i)_{10}^m) & \text{if } w_i \neq \max(w_i) \end{cases}$$

$$w_i = w_i / \sum_{i=1} w_i$$

where σ_i is the converge range factor of $f_i(x)$, whose default value is 1.0 in the paper, λ_i is the stretch factor for each $f_i(x)$, which is defined as:

$$\lambda_i = \sigma_i \cdot \frac{X_{max} - X_{min}}{x_{max}^i - x_{min}^i}$$

where $[X_{max}, X_{min}]^n$ is the search range of $F(x)$ and $[x_{max}^i, x_{min}^i]^n$ is the search range of $f_i(x)$.

In Eq. (5), $f'_i(x) = C \cdot f_i(x)/|f_{max}^i|$, where C is a predefined constant, which is set to 2000 as in [11], and f_{max}^i is the estimated maximum value of $f_i(x)$, which is estimated as:

$$f_{max}^i = f_i(x_{max} \cdot \mathbf{M}_i)$$

In the composition DBG, $\mathbf{M}$ is randomly initialized using the above transformation matrix construction algorithm and then remains unchanged. The dynamism of the system control parameter $\mathbf{H}$ and $\mathbf{O}$ can be described as follows:

$$\begin{aligned} \mathbf{H}(t+1) &= \text{DynamicChanges}(\mathbf{H}(t)) \\ \mathbf{O}(t+1) &= \text{DynamicChanges}(\mathbf{O}(t)) \end{aligned}$$

Five basic benchmark functions are used in the GDBG system. Table 1 shows the details of the five functions.

3.3 Generator Instances in the Combinatorial Space

In this section, the dynamic multi-dimensional knapsack problem (DMKP) and dynamic TSP (DTSP) are instantiated from the GDBG system.

Table 1. Details of the basic benchmark functions

name	function	range
Sphere	$f(x) = \sum_{i=1}^n x_i^2$	$[-100,100]$
Rastrigin	$f(x) = \sum_{i=1}^n (x_i^2 - 10 \cos(2\pi x_i) + 10)$	$[-5,5]$
Weierstrass	$f(x) = \sum_{i=1}^n \left(\sum_{k=0}^{k_{max}} [a^k \cos(2\pi b^k (x_i + 0.5))] \right) - n \sum_{k=0}^{k_{max}} [a^k \cos(\pi b^k)]$ $a = 0.5, b = 3, k_{max} = 20$	$[-0.5,0.5]$
Griewank	$f(x) = \frac{1}{4000} \sum_{i=1}^n (x_i)^2 - \prod_{i=1}^n \cos(\frac{x_i}{\sqrt{i}}) + 1$	$[-100,100]$
Ackley	$f(x) = -20 \exp(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2}) - \exp(\frac{1}{n} \sum_{i=1}^n \cos(2\pi x_i)) + 20 + e$	$[-32,32]$

Dynamic Multi-dimensional Knapsack Problem (DMKP)

The knapsack problem [12] is a classical combinatorial benchmark problem to test the performance of EAs. The static multi-dimensional knapsack problem (MKP) belongs to the class of NP-complete problems. It has a wide range of real world applications, such as cargo loading, selecting projects to fund, budget management, etc. The DMKP can be defined as $f(x, \phi, t)$, $\phi = (\mathbf{R}, \mathbf{P}, \mathbf{C})$ and $\mathbf{R}$, $\mathbf{P}$, and $\mathbf{C}$ are the vector of resources, profits, and capacities respectively. The DMKP can be formalized as:

$$f(x, \phi, t) = \text{Max} \sum_{i=1}^n p_i(t) \cdot x_i(t) \quad (6)$$

$$\text{subject to} \sum_{i=1}^n r_{ij}(t) \cdot x_i(t) \leq c_i(t), j = 1, 2, \dots, m \quad (7)$$

where n is the number of items, m is the number of resources, $x_i \in \{0,1\}$ indicates whether item i is included in the subset or not, p_i is the profit of item i , r_{ij} shows the resource consumption of item i for resource j , and c_i is the capacity constraint of resource i . The system control parameters are changed as:

$$\begin{aligned} \mathbf{P}(t+1) &= \text{DynamicChanges}(\mathbf{P}(t)) \\ \mathbf{C}(t+1) &= \text{DynamicChanges}(\mathbf{C}(t)) \\ \mathbf{R}(t+1) &= \text{DynamicChanges}(\mathbf{R}(t)) \end{aligned}$$

where the item profits, resources, and consumption constraints are bounded in the range of $[l_p, u_p]$, $[l_r, u_r]$, and $[l_c, u_c]$ respectively. The change severity of item profits, resources, and consumption constraints should read $\phi_{\mathbf{P}\text{-}severity}$, $\phi_{\mathbf{R}\text{-}severity}$ and $\phi_{\mathbf{C}\text{-}severity}$, all of them are set 1 in DBG system.

Dynamic Traveling Salesman Problem (DTSP)

TSP is another classical NP-complete combinatorial problem. DTSP [13] has a wide range of real applications, especially in the optimization of dynamic networks, like network planning and designing, load-balance routing, and traffic management.

DTSP is a TSP determined by a dynamic cost (distance) matrix as follows:

$$D(t) = \{d_{ij}(t)\}_{n \times n} \quad (8)$$

where $d_{ij}(t)$ is the cost from city i to city j , n is the number of cities. DTSP can be defined as $f(x, \phi, t)$, $\phi = \mathbf{D}$, the objective of DTSP is to find a minimum-cost route containing all cities at time t . It can be described as:

$$f(x, \phi, t) = \text{Min}(\sum_{i=1}^n d_{T_i, T_{i+1}}(t)) \quad (9)$$

where $T \in 1, 2, \dots, n$, if $i \neq j$, then $T_i \neq T_j$, $T_{n+1} = T_1$. The dynamism of the cost matrix $\mathbf{D}$ is described as:

$$\mathbf{D}(t+1) = \text{DynamicChanges}(\mathbf{D}(t)).$$

4 Experimental Study

In this section, some experiments based on PSO and FEP [9] algorithms are carried out on the two instances of real space to test the performance of the GDBG system. The number of dimensions $n = 10$, peak number and basic function number $m = 10$ in both instances, the 10 basic functions selected are all sphere function in the composition DBG, the search range is set to $x \in [-5, 5]$ for both DBGs. $\text{noisy}_{\text{severity}} = 0.5$, Chaotic constant $A = 3.57$, Width severity $\phi_{\text{w}_{\text{severity}}} = 0.5$. The population size is set to 50 for both PSO and FEP. For the PSO algorithm, acceleration constants η_1 and η_2 are both set to be 1.496180 and the inertia weight $\omega = 0.729844$. In FEP, the tournament size is 5 for selection and the initial standard deviation is 3.0 as used in [9]. Both algorithms were run 30 times independently for all the results.

For evaluating the efficiency of the algorithms, we use the offline performance measure defined as follows:

$$e_t = \frac{h_t}{f_t}, \quad (10)$$

where f_t is the best solution got by an algorithm just before the t -th environmental change, h_t is the optimum value at time t , and e_t is the relative value of h_t and f_t . Fig. 2 shows the offline performance of PSO and FEP in the composition DBG and rotation DBG environments.

From Fig. 2, it can be seen that algorithms show a different performance in different dynamic types. Algorithms give a much better performance in small step changes than large step changes. Both PSO and FEP show the recurrent performance in the recurrent and recurrent with noisy environments. On the other hand, the composition DBG environment is much harder to search than the rotation DBG environment for all change types.

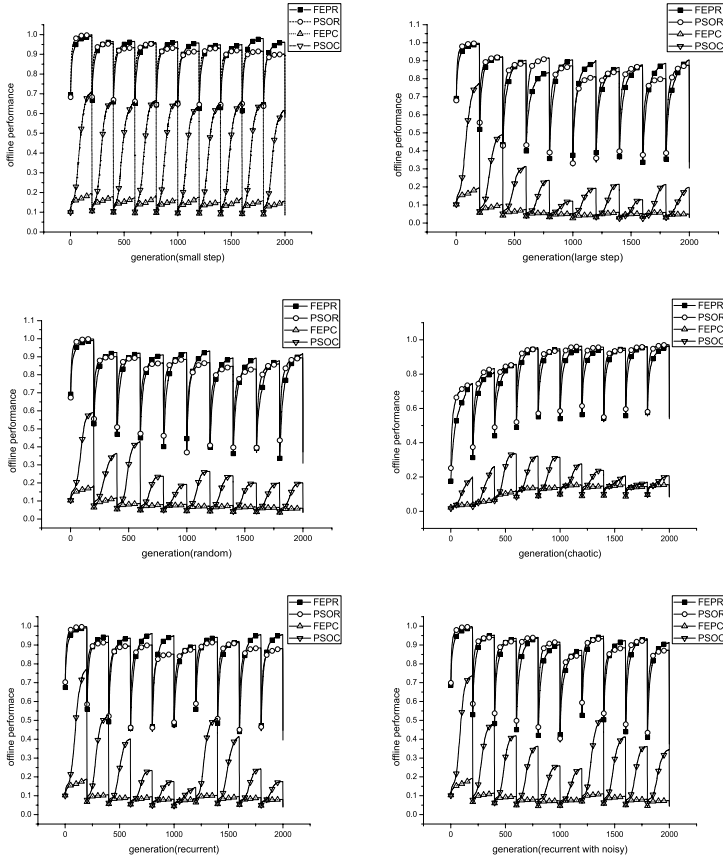


Fig. 2. The offline performance of PSO and FEP in the composition and rotation DBG environments

5 Conclusions

Constructing benchmark problems is an important task in studying EAs in dynamic environments. This paper proposes a unified method, GDBG, to construct dynamic environments across the binary, real, and combinatorial solution spaces. GDBG introduces a rotation method instead of shifting peak positions as in the MPB. The rotation method can overcome the shortcomings of unequal challenge per change for algorithms of the MPB model, which occurs when the peak positions bounce back from the boundary of the landscape. In order to test the GDBG system, experiments were carried out using the PSO and FEP algorithms under the composition DBG and rotation DBG environments. Experimental results show that the GDBG system can give different properties by simply setting the environment change type.

Though GDBG system can give different properties, there's lack of real-world application model. In the future, we will add more real-world related models and extend more dynamic instances.

Acknowledgement

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of UK under Grant EP/E060722/1.

References

1. Lewis, J., Hart, E., Ritchie, G.: A comparison of dominance mechanisms and simple mutation on non-stationary problems. In: Proc. of the 5th Int. Conf. on Parallel Problem Solving from Nature, pp. 139–148 (1998)
2. Branke, J.: Memory enhanced evolutionary algorithms for changing optimization problems. In: Proc. of the 1999 Congr. on Evol. Comput., pp. 1875–1882 (1999)
3. Morrison, R.W., De Jong, K.A.: A test problem generator for non-stationary environments. In: Proc. of the 1999 Congr. on Evol. Comput., pp. 2047–2053 (1999)
4. Yang, S.: Non-stationary problem optimization using the primal-dual genetic algorithm. In: Proc. of the 2003 IEEE Congr. on Evol. Comput., pp. 2246–2253 (2003)
5. Yang, S., Yao, X.: Experimental study on population-based incremental learning algorithms for dynamic optimization problems. *Soft Comput.* 9(11), 815–834 (2005)
6. Yang, S., Yao, X.: Population-based incremental learning with associative memory for dynamic environments. *IEEE Trans. on Evol. Comput.* (2008)
7. Eberhart, R.C., Kennedy, J.: A new optimizer using particle swarm theory. In: Proc. of the 6th Int. Symp. on Micro Machine and Human Science, pp. 39–43 (1995)
8. Kennedy, J., Eberhart, R.C.: Particle Swarm Optimization. In: Proc. of the 1995 IEEE Int. Conf. on Neural Networks, pp. 1942–1948 (1995)
9. Yao, X., Liu, Y.: Fast evolutionary programming. In: Proc. of the 5th Annual Conf. on Evolutionary Programming, pp. 451–460 (1996)
10. Weicker, K., Weicker, N.: Dynamic rotation and partial visibility. In: Proc. of the IEEE 2003 Congr. on Evol. Comput., pp. 1125–1131 (2003)
11. Liang, J.J., Suganthan, P.N., Deb, K.: Novel composition test functions for numerical global optimization. In: Proc. of the 2005 IEEE Congr. on Evol. Comput., pp. 68–75 (2005)
12. Kellerer, H., Pferschy, U., Pisinger, D.: *Knapsack Problems*. Springer, Heidelberg (2004)
13. Li, C., Yang, M., Kang, L.: A new approach to solving dynamic TSP. In: Wang, T.-D., Li, X.-D., Chen, S.-H., Wang, X., Abbass, H.A., Iba, H., Chen, G.-L., Yao, X. (eds.) SEAL 2006. LNCS, vol. 4247, pp. 236–243. Springer, Heidelberg (2006)

A Study on the Performance of Substitute Distance Based Approaches for Evolutionary Many Objective Optimization

Hemant K. Singh, Amitay Isaacs, Tapabrata Ray, and Warren Smith

School of Aerospace, Civil and Mechanical Engineering,
University of New South Wales, Australian Defence Force Academy, Canberra, ACT
{h.singh,a.isaacs,t.ray,w.smith}@adfa.edu.au

Abstract. Non-dominated Sorting Genetic Algorithm (NSGA-II) [1] and the Strength Pareto Evolutionary Algorithm (SPEA2) [2] are the two most widely used evolutionary multi-objective optimization algorithms. Although, they have been quite successful so far in solving a wide variety of real life optimization problems mostly 2 or 3 objective in nature, their performance is known to deteriorate significantly with an increasing number of objectives. The term *many objective optimization* refers to problems with number of objectives significantly larger than two or three. In this paper, we provide an overview of the challenges involved in solving many objective optimization problems and provide an in depth study on the performance of recently proposed substitute distance based approaches, viz. Subvector dominance, -eps-dominance, Fuzzy Pareto Dominance and Sub-objective dominance count for NSGA-II to deal with many objective optimization problems. The present study has been conducted on scalable benchmark functions (DTLZ2-DTLZ3) and the recently proposed P* problem [3] since their convergence and diversity measures can be compared conveniently. An alternative substitute distance approach is introduced in this paper and compared with existing ones on the set of benchmark problems.

1 Introduction

Multi-objective optimization has traditionally focused mostly on 2 or 3 objective problems. Evolutionary multi-objective algorithms such as NSGA-II [1] and SPEA2 [2] have had an outstanding success in dealing with multi-objective optimization problems in recent past. Both the algorithms rely on Pareto dominance to rank solutions, which unfortunately weakens as the number of objectives increase. The deterioration of their performance with an increasing number of objectives have already been identified and reported in literature [4,5].

The number of non-dominated solutions in a randomly initialized population of 100 individuals for DTLZ1-DTLZ4 is presented in Figure 1(a). It is clear from Figure 1(a), that 70% or more solutions of 20 objective DTLZ1- DTLZ4 problems are non-dominated at the initialization stage.

The challenges involved in dealing with many objective optimization problems can be classified into two major categories (a) deterioration of the search ability of the algorithm due to the ineffectiveness of Pareto dominance (b) exponential increase in

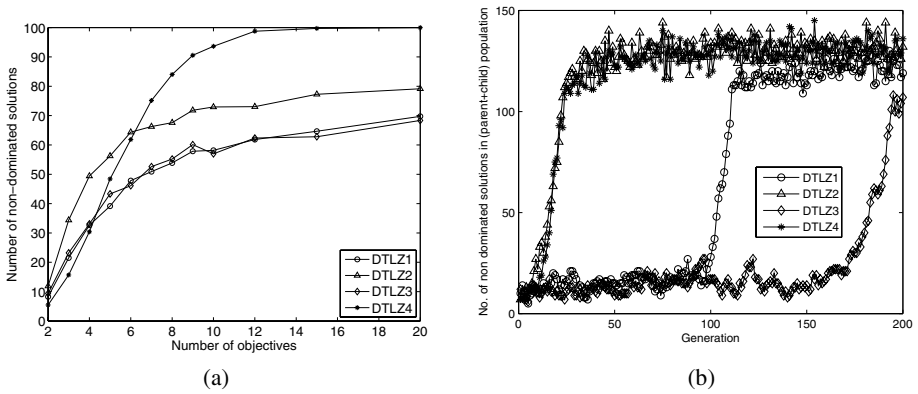


Fig. 1. (a) Number of non-dominated solutions in a randomly initialized population for DTLZ, (b) Number of non-dominated (rank 1) solutions for bi-objective DTLZ problems in (parent+child) population with generations during a typical run of NSGA-II. Population size used in 100.

the number of solutions required to approximate the Pareto surface with a large number of objectives.

In order to improve the search ability (convergence) of an algorithm, mechanisms are required to modify the ranking process to create a selection pressure to the drive solutions to the Pareto front. Proposals to use average ranking [6], expand or shrink dominance relations [7] have been suggested in literature. The ranking can also be based on scalar functions (weighted aggregation of objectives as in Multi-objective Genetic Local Search [8,9,10]) or indicator based such as hypervolume as suggested in [11]. Weighted aggregation, average rank and the expansion dominance relation based schemes are known to decrease the diversity of the solutions [4], while the indicator based approaches (e.g. hypervolume) are known to be computationally expensive.

The number of solutions required to capture the entire Pareto surface for many objective optimization problems grows exponentially with the number of objectives. It effectively means, in order to capture solutions spread across the entire Pareto set, one needs to use a larger population size. However, for many practical problems, a user is often interested in solutions around certain regions of interest. Proposals based on reference point methods [12], preference articulation method [13] and incorporation of reference point preference schemes proposed in [14] fall into this category. The inherent problems of multidimensional data visualization comes into play for many objective optimization problems. Proposals to view multidimensional data range from the use of Self Organizing Maps [15], Parallel Plots [13], Heatmaps [16] and Web diagrams [17]. The use of dimensionality reduction has also been proposed by Saxena and Deb [18] for certain class of problems to reduce and transform the original set of objectives to a lower dimensional space.

It is clear from the above discussion that there is a significant recent interest in developing or improving existing algorithms to deal with many objective optimization problems. Substitute distance arrangement mechanisms have been recently proposed by Koppen and Yoshida [3] to be used in lieu of crowding distance in NSGA-II for

many objective optimization problems. In this paper, we investigate them in greater detail and propose an alternative substitute distance measure based on clustering and compare our results.

Before getting into the details of the proposal, it is necessary to highlight the existing mechanisms within NSGA-II. In NSGA-II, the parents and their children are ranked via a primary ranking scheme based on non-dominance and the solutions sharing the same rank are subjected to secondary ranking based on their crowding distance. The crowding distance is a mechanism to spread the solutions in the objective space which is useful once the algorithm has converged to the Pareto front. Since the majority of solutions are non-dominated for many objective problems even at the initialization stage, NSGA-II essentially ranks them based on their crowding distance. It is also worth noting that the crowding distance mechanism in NSGA-II assigns an infinite fitness to solutions which have the best objective function value in any one of the objectives which by virtue of elitism ensures their presence in the final population.

For bi-objective DTLZ problems, a plot of number of non-dominated solutions (for a population size of 100) in (parent+child) solutions with generations is shown in Figure 1(b). It is interesting to observe that even for a bi-objective problem, a typical MOEA run is essentially driven by secondary ranking beyond certain generations, *i.e.* when the number of non-dominated solutions become greater than the population size (after 30 generations for DTLZ2 & DTLZ4, 100 generations for DTLZ1 and 200 generations for DTLZ3). The above observation is significant as an early reliance solely on secondary ranking could lead to poor convergence even for bi-objective optimization problems. The convergence may slow down for cases where all points in the population are non-dominated but still far from the Pareto front, because the algorithm would be driven mainly by the diversity preserving mechanism instead of non-dominated sorting.

The rest of the paper is organized as follows. A description of various substitute distance assignments for NSGA-II to handle many objective problems, is given in Section 2. Test problems studied are described in Section 3, followed by numerical experiments in Section 4. Finally, the summary and conclusions of the present studies are given in Section 5.

2 Secondary Ranking Assignments

The reason for poor performance of evolutionary algorithms for large number of objectives is attributed to the loss of selection pressure owing to the significant increase in the proportion of non-dominated solutions in the population. To overcome this drawback, a number of substitute distance assignment measures have been suggested recently [3], which can be used instead of crowding distance in NSGA-II in order to improve convergence. The idea is to devise measures that would differentiate amongst the non-dominated solutions, thus creating a selection pressure and improving the convergence rate towards the Pareto-front. A brief description of substitute distance assignments suggested in [3] are given below, followed by another scheme which we propose here.

2.1 Subvector Dominance (SV-DOM)

Subvector dominance counts the number of objectives in which a solution is better than other solutions. A solution better in more number of objectives than others, is hence considered better. For a given solution i in the population, procedure $svd(i, j)$ counts the number of objectives another solution j is better than i . The values $svd(i, j)$ are calculated for all $j \neq i$, and largest such value is assigned as the distance $dist(i)$ to the solution i . Smaller the value of $dist(i)$, better is the solution.

2.2 ϵ -dominance (ϵ -DOM)

ϵ -dominance ranks a given solution based on the smallest amount that should be subtracted from all objectives of other solutions to make them dominate the given solution. For a solution i , the value $mepsd(i, j)$ denotes the smallest amount to be subtracted from all the objectives of solution j , so that it dominates solution i . Smallest such value among all solutions $j \neq i$ is the distance $dist(i)$ assigned to the solution i . A larger $dist(i)$ value implies better solution, as it means that a large amount has to be subtracted from the objectives of other solutions to make them dominate the given solution.

2.3 Fuzzy Pareto Dominance (FPD)

FPD assignment is based on Fuzzy-Pareto-Dominance relation as presented in [19]. For a solution i in the population, the product of bounded quotients $F(i).m/F(j).m$ of all objectives is calculated for all $j \neq i$ (where $F(i).m$ represents m^{th} objective value for i^{th} solution). The largest value of the product among all other solutions $j \neq i$ is assigned to i as the distance value. Smaller $dist(i)$ implies a better solution.

2.4 Sub-objective Dominance Count (SOD-CNT)

SOD-CNT ranks the solutions based on two separate rankings: SV-DOM and ϵ -DOM. For each solutions i in the non-dominated set, a set S_i is constructed as all pairs of two single criterion distance measures: $M - svd(i, j)$ (where M is the number of objectives), and $mepsd(i, j)$, for all $j \neq i$. The number of solutions in the Pareto set PSO_i of the set S_i is assigned as $dist(i)$ to the solution. Larger $dist(i)$ implies better solution.

2.5 Cluster-Sort

Apart from the above mentioned distance assignments described above, another substitute distance assignment is suggested here, and is referred to as Cluster-sort. The method can be used as an alternate scheme to preserve diversity. The proposed method doesn't show any bias to the corner points, unlike NSGA-II. The preference for the corner solutions may be disadvantageous for many objective problems, as a large number of solutions in the front may be "extreme" points in one or more of the objectives. In such cases, the distance assigned using crowding distance to each of those points would be infinity, and hence there would be no preferred point amongst them. The drawback of maintaining corner solutions has also been previously highlighted in [20].

The Cluster-sort distance assignment is shown in Algorithm 1. The non-dominated front F to be ranked is first clustered into half the number of points it contains, i.e. $|F|/2$. (if $|F|$ is odd, then $\text{ceiling}(|F|/2)$ is used). Then within each cluster, a point closest to its centroid is identified. All such closest points to the centroids are assigned lower rank than the rest of the solutions in F . The procedure is shown in Algorithm 1. For the present studies, Hierarchical clustering has been used. The motivation behind clustering into $|F|/2$ points is so that during the run, when the (parent+child) population is largely non-dominated (rank 1), points with good diversity could be chosen as the parent population for the next generation.

Algorithm 1. Cluster-sort

Require: Front data $\{\text{non-dominated rank, corresponding set } F\}$

```

1: for  $i = 1$  to  $|F|$  do
2:   Assign  $\text{dist}(i) = \text{non-dominated rank} + 0.5$ .
3: end for
4:  $A = \text{cluster}(F \rightarrow |F|/2)$  {cluster set  $F$  into  $|F|/2$  points}
5:  $C \leftarrow \text{Centroids}(A)$ 
6: for  $i = 1$  to  $|F|/2$  do
7:    $\text{id} = \min(\text{norm}(x_{\text{id} \in C(i)} - C(i)))$  {Find id of closest point to  $i^{\text{th}}$  cluster}
8:   Assign  $\text{dist}(\text{id}) = \text{non-dominated rank}$ . {The points closest to centroids get smaller  $\text{dist}$ }
9: end for
10: {Smaller  $\text{dist} \Rightarrow$  Better solution.}
  
```

3 Test Problems

Studies on two sets of test problems are presented in this paper:

3.1 DTLZ Problems

DTLZ is a set of scalable multi-objective test problems [21]. The problems are scalable in terms of the number of objectives as well as the number of variables. Problems DTLZ2 and DTLZ3 have been used for the present study, for up to 25 objectives.

3.2 P* Problems

A class of problems was developed by Koppen and Yoshida [3] for ease of visualization of Pareto-front coverage, which they refer to as P* problems. The problem is as follows. Given a set of m fixed points $(P_1, P_2, \dots, P_m)$ in Euclidean place, the objective values $f_1, f_2, \dots, f_m$ at a given point x_i are $d(x_i, P_1), d(x_i, P_2), \dots, d(x_i, P_m)$ respectively, where $d(A, B)$ denotes the Euclidean distance between the two points A and B. The aim is to minimize all the objectives $f_1, f_2, \dots, f_m$.

The Pareto set of the problem is given by the convex enclosure of the points P_i . The proof and a detailed discussion of the same, can be found in [17]. Studies P* problems up to 20 objectives have been presented in this paper.

4 Numerical Experiments

Numerical experiments were carried out on DTLZ2 and DTLZ3 and the P* problem to investigate the performance of various substitute distance assignment measures in terms of convergence and Pareto front coverage.

For DTLZ2 and DTLZ3, twenty independent runs were made for 5, 10, 15, 20 and 25 objectives problem using all secondary distance assignment strategies (Crowding distance, SV-DOM, $-\epsilon$ -DOM, FPD, SOD-CNT, and Cluster-sort). The following parameters were used: probability of crossover = 1.0, probability of mutation = $1/n$ (where n is the number of variables), crossover index = 15, and mutation index = 20. These parameter settings are same as used in [21]. A population of 100 solutions was evolved for 200 generations for each run. For DTLZ2 and DTLZ3, the convergence measure for a solution can be found using $\left(\sqrt{\sum_{i=1}^M f_i^2} - 1\right)$, where M is the number of objectives. The measure was calculated for each point in the final population and the mean was used as the convergence metric. The convergence metrics averaged over all runs using various strategies for DTLZ2 and DTLZ3 are listed in Table 1 and 2 respectively. From the Table 1, it is seen that for the case of DTLZ2, Cluster-sort compares poorly with the recently proposed substituted distance assignments, but it is able to achieve better convergence than the default crowding distance measure. For DTLZ3, however, Cluster-sort performs worse than crowding distance. The convergence of $-\epsilon$ -DOM, SV-DOM and SOD-CNT are excellent for both DTLZ2 and DTLZ3. FPD also performs significantly better than both Cluster-sort and the crowding distance.

Apart from the convergence to the Pareto front, a good diversity of solutions is also desirable for most multi-objective problems. For the ease of visualization of the diversity of obtained points, P* problem [3] was studied using various strategies. The diversity of points can be seen in the variable space as suggested in [3]. As mentioned earlier, the Pareto optimal set for the problem as seen in variable space is the convex enclosure of the set of points used for calculating objectives.

For P* problem, the fixed points used to calculate objectives were chosen as the corners of a M sided regular polygon in the variable (2-dimensional) space, where M is the number of objectives (see Figure 2). The parameters used were: probability of crossover = 0.9, probability of mutation = 0.1, crossover index = 15, and mutation

Table 1. DTLZ2 average convergence metric

Strategy	5 - obj		10 - obj		15 - obj		20 - obj		25 - obj	
	Mean	S.D.	Mean	S.D.	Mean	S.D.	Mean	S.D.	Mean	S.D.
CROW-DIST	0.067	0.016	1.864	0.080	2.032	0.051	2.077	0.054	2.094	0.078
CLUSTER-SORT	0.056	0.016	1.715	0.072	1.901	0.050	1.963	0.070	1.965	0.078
SV-DOM	0.014	0.006	0.063	0.024	0.058	0.055	0.003	0.002	0.003	0.002
$-\epsilon$ -DOM	0.018	0.002	0.042	0.006	0.057	0.011	0.060	0.017	0.065	0.022
FPD	0.027	0.007	0.193	0.081	0.267	0.077	0.279	0.086	0.301	0.093
SOD-CNT	0.004	0.001	0.003	0.000	0.003	0.001	0.002	0.001	0.002	0.001

Table 2. DTLZ3 average convergence metric

Strategy	5 - obj		10 - obj		15 - obj		20 - obj		25 - obj	
	Mean	S.D.	Mean	S.D.	Mean	S.D.	Mean	S.D.	Mean	S.D.
CROW-DIST	441.26	46.95	1395.82	83.34	1635.29	56.57	1705.55	53.43	1727.44	49.42
CLUSTER-SORT	719.12	67.44	1585.90	44.72	1697.47	50.32	1735.60	63.04	1756.76	52.40
SV-DOM	161.29	30.03	215.12	68.89	129.75	65.93	80.64	39.94	57.78	24.67
- ϵ -DOM	3.22	1.72	5.17	5.52	8.56	6.47	9.03	4.80	9.42	6.96
FPD	200.57	53.16	308.29	64.37	345.68	107.71	301.74	93.88	325.53	90.68
SOD-CNT	15.54	6.67	22.79	12.86	91.55	124.76	126.03	119.81	271.00	298.17

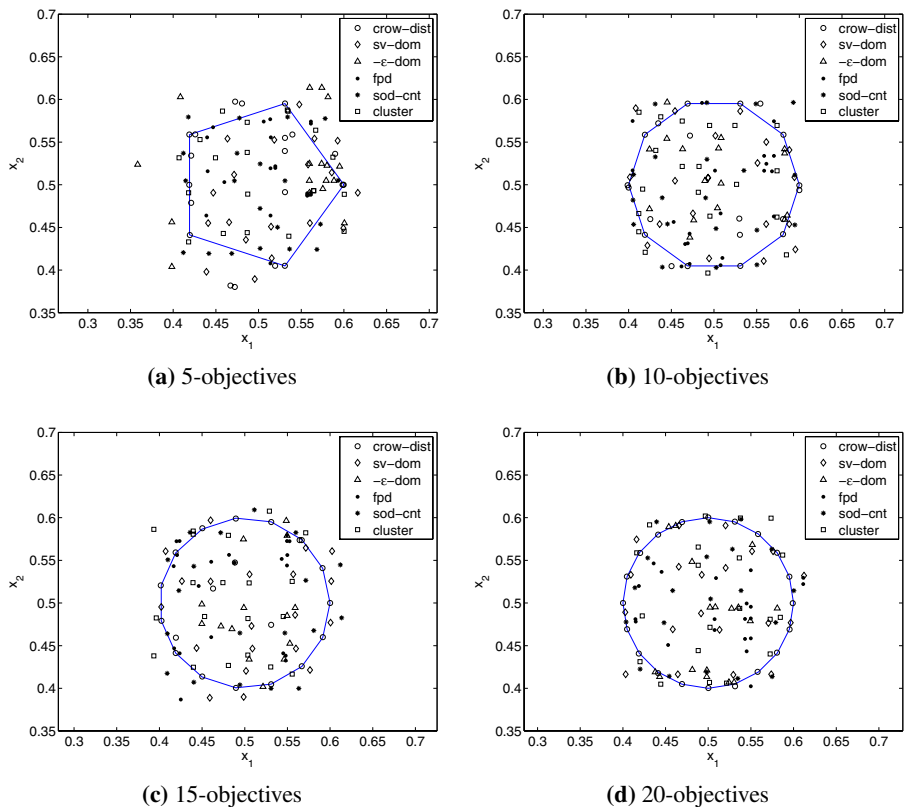


Fig. 2. The distribution of final population obtained for P^* problem in variable space using various distance assignments with NSGA-II

index = 20. A population of 20 solutions was evolved over 100 generations. Studies were made for 5, 10, 15 and 20 objectives.

The final populations from a typical run using each of the strategies is shown in Figure 2. It is seen that the points obtained by Cluster-sort are well spread, showing

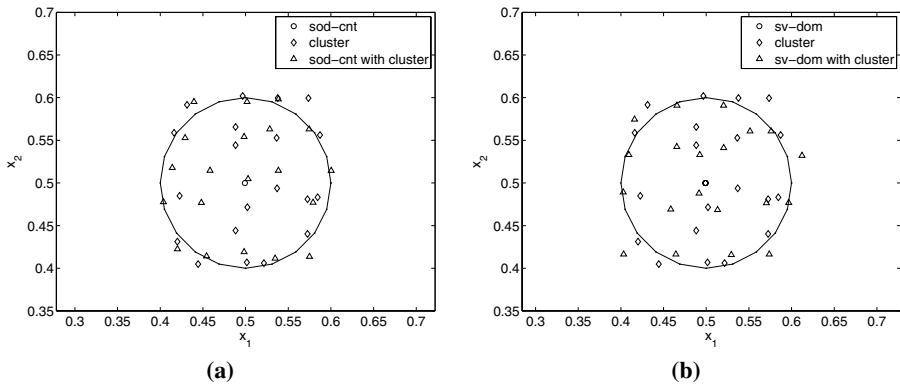


Fig. 3. The distribution of final population obtained for 20 objective P^* problem in variable space using combination of schemes: (a) SV-DOM with Cluster-sort, (b) SOD-CNT with Cluster-sort

better diversity than rest of the methods. Crowding distance, due to its preference for extreme points, obtains solutions in the periphery of the polygon (also observed in [3]). SV-DOM and SOD-CNT show good convergence, but show very poor diversity. $-\epsilon$ -DOM and FPD show a good tradeoff between convergence and diversity.

From the preceding discussion, it is clear that while the convergence of Cluster-sort is not good compared to other approaches, but it obtains a good diversity among the solutions. Hence, it would be more suitable to use this scheme in conjunction with other fast convergence schemes (with poor diversity). Studies were done for 20-objective P^* problem by using a combination of Cluster-sort with SV-DOM and SOD-CNT. The parameters used were the same as those used in earlier P^* problem studies. Results obtained using SOD-CNT for first 40 generations and Cluster-sort thereafter (up to 100 generations) is shown in Figure 3(a). Figure 3(b) shows the results obtained using SV-DOM for first 40 generations and Cluster-sort thereafter. Also shown are the results obtained using the schemes individually. It can be seen that if SV-DOM or SOD-CNT alone are used, then the solutions show poor diversity. On the other hand, if Cluster-sort alone is used, then the convergence is not good, which is reflected in the presence of quite a few solutions outside the 20-sided polygon (as the solutions inside the polygon are Pareto-optimal). However, when they are used in combination, the solutions have better convergence (less solutions outside the polygon). At the same time, the diversity of the solutions is also good.

5 Summary and Conclusions

In this paper, an overview of challenges involved in many-objective optimization is given. The performance of various recently proposed secondary ranking assignment methods instead of crowding distance in NSGA-II is investigated. An additional secondary distance assignment measure (Cluster-sort) is proposed and its performance was studied in terms of convergence and diversity. It is observed that though the proposed approach does not accelerate convergence, but it is able to achieve a good diversity

among the population points. It might be a good idea to use the proposed technique in conjunction with approaches with faster convergence. A few preliminary studies to the effect are provided, which show encouraging results.

Acknowledgments

The presented work was supported by grants from Defence and Security Applications Research Center (DSARC) at UNSW@ADFA, Canberra, Australia.

References

1. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on Evolutionary Computation* 6, 182–197 (2002)
2. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: improving the strength pareto evolutionary algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland (2002)
3. Koppen, M., Yoshida, K.: Substitute distance assignments in NSGA-II for handling many-objective optimization problems. In: Obayashi, S., Deb, K., Poloni, C., Hiroyasu, T., Murata, T. (eds.) EMO 2007. LNCS, vol. 4403, pp. 727–741. Springer, Heidelberg (2007)
4. Ishibuchi, H., Tsukamoto, N., Nojima, Y.: Evolutionary many-objective optimization. In: 3rd International Workshop on Genetic and Evolving Systems (GEFS 2008), pp. 47–52 (March 2008)
5. Khare, V., Yao, X., Deb, K.: Performance Scaling of Multi-objective Evolutionary Algorithms. In: Fonseca, C.M., Fleming, P.J., Zitzler, E., Deb, K., Thiele, L. (eds.) EMO 2003. LNCS, vol. 2632, pp. 376–390. Springer, Heidelberg (2003)
6. Corne, D.W., Knowles, J.D.: Techniques for highly multiobjective optimisation: some non-dominated points are better than others. In: Proceedings of the 9th annual conference on Genetic and evolutionary computation (GECCO 2007), pp. 773–780. ACM, New York (2007)
7. Sato, H., Aguirre, H., Tanaka, K.: Controlling dominance area of solutions and its impact on the performance of moeas. In: Obayashi, S., Deb, K., Poloni, C., Hiroyasu, T., Murata, T. (eds.) EMO 2007. LNCS, vol. 4403, pp. 5–20. Springer, Heidelberg (2007)
8. Ishibuchi, H., Murata, T.: A multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 28(3), 392–403 (1998)
9. Ishibuchi, H., Yoshida, T., Murata, T.: Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. *IEEE Transactions on Evolutionary Computation* 7(2), 204–223 (2003)
10. Jaszkievicz, A.: Genetic local search for multi-objective combinatorial optimization. *European Journal of Operational Research* 127(1), 50–71 (2002)
11. Zitzler, E., Kunzli, S.: Indicator-Based Selection in Multiobjective Search. In: Yao, X., Burke, E.K., Lozano, J.A., Smith, J., Merelo-Guervós, J.J., Bullinaria, J.A., Rowe, J.E., Tiño, P., Kabán, A., Schwefel, H.-P. (eds.) PPSN 2004. LNCS, vol. 3242, pp. 832–842. Springer, Heidelberg (2004)
12. Deb, K., Sundar, J.: Reference point based multi-objective optimization using evolutionary algorithms. In: Proceedings of the 8th annual conference on Genetic and evolutionary computation (GECCO 2006), pp. 635–642. ACM, New York (2006)
13. Fleming, P., Purshouse, R., Lygoe, R.: Many-Objective Optimization: An Engineering Design Perspective. In: Coello Coello, C.A., Hernández Aguirre, A., Zitzler, E. (eds.) EMO 2005. LNCS, vol. 3410, pp. 14–32. Springer, Heidelberg (2005)

14. Thiele, L., Miettinen, K., Korhonen, P., Molina, J.: A preference-based interactive evolutionary algorithm for multiobjective optimization. Technical Report W-412, Helsinki School of Economics (2007)
15. Obayashi, S., Sasaki, D.: Visualization and data mining of pareto solutions using self-organizing map. In: Fonseca, C.M., Fleming, P.J., Zitzler, E., Deb, K., Thiele, L. (eds.) EMO 2003. LNCS, vol. 2632, pp. 796–809. Springer, Heidelberg (2003)
16. Pryke, A., Sanaz Mostaghim, A.N.: Heatmap visualization of population based multi objective algorithms. In: Obayashi, S., Deb, K., Poloni, C., Hiroyasu, T., Murata, T. (eds.) EMO 2007. LNCS, vol. 4403, pp. 361–375. Springer, Heidelberg (2007)
17. Koppen, M., Yoshida, K.: Many-objective particle swarm optimization by gradual leader selection. In: Beliczynski, B., Dzielinski, A., Iwanowski, M., Ribeiro, B. (eds.) ICANNGA 2007. LNCS, vol. 4431, pp. 323–331. Springer, Heidelberg (2007)
18. Saxena, D.K., Deb, K.: Trading on infeasibility by exploiting constraints criticality through multi-objectivization: A system design perspective. In: Proceedings of IEEE Congress on Evolutionary Computation (CEC 2007), September 25–28, 2007, pp. 919–926 (2007)
19. Koppen, M., Vincente-Garcia, R., Nickolay, B.: Fuzzy-pareto-dominance and its application in evolutionary multi-objective optimization. In: Fonseca, C.M., Fleming, P.J., Zitzler, E., Deb, K., Thiele, L. (eds.) EMO 2003. LNCS, vol. 2632, pp. 399–412. Springer, Heidelberg (2003)
20. Wagner, T., Beume, N., Naujoks, B.: Pareto-, aggregation-, and indicator-based methods in many-objective optimization. In: Obayashi, S., Deb, K., Poloni, C., Hiroyasu, T., Murata, T. (eds.) EMO 2007. LNCS, vol. 4403, pp. 742–756. Springer, Heidelberg (2007)
21. Deb, K., Thiele, L., Laumanns, M., Zitzler, E.: Scalable multi-objective optimization test problems. In: Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002), vol. 1, pp. 825–830 (May 2002)

Performance Evaluation of an Adaptive Ant Colony Optimization Applied to Single Machine Scheduling

Davide Anghinolfi¹, Antonio Boccalatte¹, Massimo Paolucci¹,
and Christian Vecchiola²

¹ Department of Communication, Computer and Systems Sciences,
University of Genova, Via Opera Pia 13, 16145 Genova, Italy
{anghinolfi,nino,paolucci}@dist.unige.it

² Department of Computer Science and Software Engineering,
The University of Melbourne, 111 Barry St, 3053, Carlton, Victoria, Australia
csve@csse.unimelb.edu.au

Abstract. We propose a self-adaptive Ant Colony Optimization (AD-ACO) approach that exploits a parameter adaptation mechanism to reduce the requirement of a preliminary parameter tuning. The proposed AD-ACO is based on an ACO algorithm adopting a pheromone model with a new global pheromone update mechanism. We applied this algorithm to the *single machine total weighted tardiness scheduling problem with sequence-dependent setup times* and we executed an experimental campaign on a benchmark available in literature. Results, compared with the ones produced by the ACO algorithm without adaptation mechanism and with those obtained by recently proposed metaheuristic algorithms for the same problem, highlight the quality of the proposed approach.

Keywords: Ant Colony Optimization, Metaheuristics, Scheduling.

1 Introduction

Ant Colony Optimization (ACO) is a metaheuristic approach which mimics the successful behaviour of ants in cooperating to find shortest paths to food for solving combinatorial problems [1]. In order to communicate real ants use stigmergy, which is a coordination mechanism using the environment as a primary mean for information exchange: a natural essence called pheromone is left by the ants along the most promising trail to the food. The pheromone trail evaporates over time and it can either disappear on the paths abandoned by the ants or be reinforced by the passage of further ants. In this way, effective (i.e., shortest) paths leading to food are finally marked by a strong pheromone trail, and they are followed by most of the ants. The ACO metaheuristic, first introduced by Dorigo [2], combines both Reinforcement Learning (RL) [3] and Swarm Intelligence (SI) [4] concepts: the agents (the ants) take decisions and receive a reward from the environment which depends on the quality of the solution found (RL); the agents exchange information to share experiences and the performance of the overall system (the colony) emerges from the collection of the simple agents' interactions and actions (SI). ACO has been successfully applied to several combinatorial optimization problems, from the travelling salesman problem

applications to vehicle routing problems, and to single machine and flow shop scheduling problems (see Dorigo and Stützle [1] as reference for ACO applications).

In this paper we propose a new Adaptive Ant Colony Optimization (AD-ACO) approach aiming at reducing the number of parameters that must be fixed by using runtime parameter self-tuning mechanism. In addition, the non-adaptive ACO algorithm which AD-ACO is based on mainly differs from previous ones in literature for several characteristics: (a) the pheromone values are independent of the problem cost (or quality) function and they are bounded within an arbitrarily chosen and fixed interval; (b) a new *global pheromone update* (GPU) rule is adopted which makes the pheromone values asymptotically increase (decrease) towards the upper (lower) bound without requiring any explicit cut-off (differently from the Max-Min Ant System (MMAS) [5]); (c) a diversification strategy is used based on a temporary perturbation of the pheromone values performed by a *local pheromone update* (LPU) rule within any single iteration. This work is an evolution of the approach recently presented in Anghinolfi and Paolucci [6] where the main features of the non-adaptative ACO algorithm here extended to AD-ACO were first introduced. In this paper we first summarize the most relevant and fundamental aspects of the non-adaptative algorithm emerged from the experimental results presented in our previous work [6]; then we focus on the new parameter adaptation method introduced in AD-ACO and on the relevant experimental campaign.

We experimented and compared our ACO and AD-ACO approaches by facing one among the most important scheduling problems, i.e., the single machine total weighted tardiness scheduling with sequence-dependent setup times (STWTSDS) problem, whose characteristics are briefly introduced in Section 2. Then, the rest of the paper is organized as follows: Section 3 discusses the main aspects of the proposed ACO and AD-ACO approaches. Section 4 shows the experimental campaign performed on the benchmark set generated by Cicirello [7] for the considered problem, comparing the proposed ACO and AD-ACO also with the best known results in the literature. Finally, Section 5 draws some conclusions.

2 The Scheduling Problem Definition

The STWTSDS problem requires to schedule on a single machine a set of n independent jobs. The machine is always available and can process only one job at a time. Each job j is ready at time zero and is characterized by a processing time p_j , a due date d_j and a weight w_j . A sequence-dependent setup time s_{ij} should be waited between the consecutive processing of a pair (i, j) of jobs. The scheduling objective is the minimization of the total weighted tardiness, i.e.,

$$Z = \min \sum_{j=1}^n w_j T_j . \quad (1)$$

where job tardiness is defined as $T_j = \max(0, C_j - d_j)$, being C_j the completion time of job j . This problem, denoted as $1/s_{ij}/\sum w_j T_j$, is strongly NP-hard since it is a special case of the $1/s_{ij}/\sum T_j$ that has been proved to be strongly NP-hard by Lawler [8]. Therefore, in order to face instances of non-trivial dimension, research efforts concentrated on heuristics and more recently on metaheuristic approaches. The best constructive

approach for the STWTSDS problem is the well-known apparent tardiness cost with setups (ATCS) heuristic proposed by Lee et al. [9]; however, metaheuristic approaches, even if more time consuming, appear to dominate constructive heuristics.

The benchmark instance set for the STWTSDS defined by Cicirello [7] was recently used to compare the performance of several metaheuristic approaches. In particular, stochastic search procedures [10], genetic algorithm (GA) [11], GA, tabu search (TS) and simulated annealing (SA) [12], and ACO [13]. More recently, the best known results for this benchmark were independently improved by Cicirello [11], Lin and Ying [12] and Liao and Juan [13]. In our previous work [6] we adopted the same experimental settings as those proposed by Liao and Juan [13] to compare the two heuristics; the obtained results showed that our algorithm dominates the ACO introduced by Liao and Juan [13] but a fair comparison with the greatest number of best known results presented by Lin and Ying [12] was not produced.

3 The ACO and AD-ACO Approaches

The ACO and AD-ACO approaches proposed in this paper are mainly based on the Ant Colony System (ACS) [14] but they include several novelties as will be illustrated in the following. The main structure of our basic ACO algorithm is reported in Fig.1.

```

Initialization;
k=1;
While <termination condition not met>
{
  For each ant  $a \in A$ 
  {
    Construction of solution  $x_a^k$ ;
    Local pheromone update;
  }
  Local Search phase;
  Global pheromone update;
  k=k+1;
}

```

Fig. 1. The basic ACO algorithm

We consider a set A of m artificial ants. At each algorithm iteration k , every ant a identifies a solution x_{ka} generating a sequence σ_{ka} of the n jobs. Note that, in general, a solution x of the STWTSDS problem is represented by a sequence $\sigma(x)=(x[1], \dots, x[n])$, and the associated objective value $Z(x)$ is simply computed by assigning to each job its feasible earliest start time for that sequence. We denote with $\sigma(x[h])$ or simply $[h]$, $h=1, \dots, n$, the index of the job that in solution x is sequenced in the h -th position on the machine, e.g., $j=\sigma(x[h])=[h]$, with $j=1, \dots, n$. In addition, we call *solution components* of x_{ka} the position-job pairs (h, j) , $h, j=1, \dots, n$, determined by a sequence $\sigma(x_{ka})$ and we associate pheromone trails $\tau_k(h, j)$ with them.

The main new aspect of the proposed ACO algorithm regards the pheromone model. In particular, we consider pheromone values assigned to $\tau_k(h, j)$ independent

of the values of the objective function associated with the explored solutions including the component (h, j) . We use pheromone trails as an *arbitrary measure* of how is a “good” practice including a component solution into a sequence. The colony learning mechanism resulting from the ants’ solution space exploration increases such a measure on promising components and reduces the same on the poor quality ones, according to the reinforcement learning concept. However, differently from other ACO approaches, our learning mechanism does not need any tuning or customization phase and it can be straightforwardly applied to different application contexts. In particular, we fix an arbitrary range $[0, \tau_{max}]$ for the pheromone values, where τ_{max} is chosen independently of the specific problem or instance considered, i.e., it is not a parameter that need to be fixed. Also in MMAS [5] lower and upper bounds τ_{min} and τ_{max} are imposed for pheromone values, which are however used as cut-off thresholds. They must be appropriately selected and dynamically updated on the basis of the specific objective function values whenever a new best solution is found.

The second new feature of our ACO approach is a new GPU rule, that we called *unbiased pheromone update* (UPU) rule, imposing a smooth variation of the pheromone values within $[0, \tau_{max}]$ so that both extremes are only asymptotically reached, preserving $\tau_k(h, j)$ strictly positive. A further new aspect of the proposed algorithm corresponds to a LPU rule that, similarly to the standard ACS, induces a pheromone perturbation favouring an intra-iteration diversification mechanism, but whose scope is kept strictly local since such perturbations are restricted to each single iteration.

3.1 The ACO Algorithm Steps

In this section we provide the details for the main steps of the algorithm in Fig.1.

Initialization. The initial pheromone value for each solution component (h, j) , $h, j=1, \dots, n$, is fixed as $\tau_0(h, j)=\tau_{max}/2$. The best current solution x^* is initialized as an empty solution and the associated objective value $Z(x^*)$ is fixed to infinity. The iterations and iterations without improvement counters are set to zero.

Iteration (i): solution construction. At each iteration k each ant a generates a solution x_{ka} by a sequence construction procedure through n selection stages: first, the set of not sequenced jobs for ant a is initialized as $U_{a0}=\{1, \dots, n\}$; then, at stage $h=1, \dots, n$, the ant selects one job j from the set $U_{a,h-1}$ and inserts it in the h -th position of the partial sequence, updating $U_{ah}=U_{a,h-1}\setminus\{j\}$ so that at stage $h=n$ the sequence is complete. Similarly to the ACS, the job selection at each stage h of the construction procedure is based on the following *pseudo-random rule*: first a random number q is extracted from the uniform distribution $U[0, 1]$; then, if $q \leq q_0$, where q_0 is a parameter fixed such that $0 \leq q_0 \leq 1$, an *exploitation* job selection rule is used, otherwise an *exploration* one is adopted. The exploitation rule, directing the ants towards the best paths previously emerged, deterministically chooses the job j to be sequenced in the h -th position as

$$j = \arg \max_{u \in U_{a,h-1}} \{ \tau_k(h, j) \cdot [\eta(h, j)]^\beta \}. \quad (2)$$

Differently, the exploration rule favours the generation of new paths selecting j according to the following probability

$$p(h, j) = \frac{\tau_k(h, j) \cdot [\eta(h, j)]^\beta}{\sum_{u \in U_a^{h-1}} \tau_k(h, j) \cdot [\eta(h, j)]^\beta}. \quad (3)$$

The so-called *heuristic value* $\eta(h, j)$ associated with the solution component (h, j) is computed according to the ATCS rule [9] as for the ACO approach adopted by Liao and Juan [13]. In this way the priority of the jobs in Eq. (2), as well as their probability in Eq. (3), are also affected by the priority index provided by ATCS: this is particularly useful to drive the ant construction process during the initial iterations of the algorithm when the pheromone is equally distributed over all the possible solution components. The quantity β in equations (2) and (3) is a parameter denoting the relative importance of the heuristic value with respect to the pheromone trail one.

Iteration (ii): local pheromone update. This step corresponds to an intra-iteration diversification performed, similarly to ACS, to avoid premature convergence of the algorithm. After an ant a completed the construction of a solution x_{ka} the following LPU rule is used to reduce the likelihood of selecting the same sequence by other ants

$$\tau_k(h, j) = (1 - \rho) \cdot \tau_k(h, j) \quad \forall h = 1, \dots, n; j = \sigma(x_{ka}[h]). \quad (4)$$

The parameter ρ , fixed $0 \leq \rho \leq 1$, is the *local evaporation rate*. We force such evaporation to be strictly local since it is based on a temporarily change of the pheromone values only in the single iteration scope that we use only as a local diversification mechanism. This feature, that we called *reset of the local pheromone update* (RLPU), appears consistent with the use of pheromone trails as learning measures that can be only updated by the GPU to take into account the overall colony exploration experience. In addition we can observe that the local update provoked by Eq. (4) is stronger than the one of standard ACS as it drives pheromone values towards zero instead of τ_0 .

Local search phase. At the completion of each iteration an intensification step is executed. The algorithm executes a local search (LS) exploration following the so-called *best in iteration* (BI) rule, i.e., starting from the solution corresponding to the best solution x_k^* found in the current iteration determined as in Eq. (5).

$$x_k^* = \arg \min_{a=1, \dots, m} Z(x_{ka}). \quad (5)$$

The LS algorithm that we adopted is similar to one defined by Tasgetiren et al. [15]. This LS algorithm executes a random neighbourhood exploration allowing both an alternation of random insert and swap moves; in addition the algorithm executes a limited number of random restarts, that we fixed equal to $n/5$, as in the iterated local search. A pseudo-code of the adopted LS can be found in Anghinolfi and Paolucci [6].

Global pheromone update. The UPU rule performs a smooth update of the pheromone trails associated with a set of quality solution components neither using any cost nor quality function value. We denote with Ω_k the set of the best solution components

determined after the completion of iteration k , i.e., the components included in the best solution so far found, i.e., $\Omega_k = \{(h, j): h=1, \dots, n, j=\alpha(x^*[h])\}$. Note that these same components are used in the so-called *BS-update* rule [1]. Then, the UPU rule operates according to the following three steps.

1. Pheromone evaporation for the solution components not included in Ω_k

$$\tau_{k+1}(h, j) = (1 - \alpha) \cdot \tau_k(h, j) \quad \forall (h, j) \notin \Omega_k. \quad (6)$$

where $0 \leq \alpha \leq 1$ is the *evaporation rate* parameter.

2. Computation of the maximum pheromone reinforcement $\Delta\tau_k(h, j)$ for the components in Ω_k

$$\Delta\tau_k(h, j) = \tau_{\max} - \tau_k(h, j) \quad \forall (h, j) \in \Omega_k. \quad (7)$$

3. Update of the pheromone trails to be used in the next iteration for the solution components in Ω_k

$$\tau_{k+1}(h, j) = \tau_k(h, j) + \alpha \cdot \Delta\tau_k(h, j) \quad \forall (h, j) \in \Omega_k. \quad (8)$$

The UPU rule guarantees that $\tau_k(h, j) \in [0, \tau_{\max}]$ and converges towards the bounds asymptotically: in fact the difference between $\tau_{k+1}(h, j)$ and $\tau_k(h, j)$ is progressively reduced as $\tau_k(h, j)$ becomes more and more close to 0 (6) or $\tau_{\max}$ (8) with a law similar to the exponential cooling schedule of the SA metaheuristic.

Termination conditions. The algorithm can be stopped when a maximum number of iterations or a maximum number of iterations without improvements is reached.

3.2 The AD-ACO Parameter Adaptation Mechanism

We implemented and tested an adaptation mechanism in our AD-ACO to provide the basic algorithm with a parameter self-tuning capability. In particular, we applied this method on two parameters, q_0 and β since we believe that their variation could provide the algorithm with better exploration flexibility.

The idea is that the algorithm should adjust the value of these parameters during the iterations, trying for them some alternative configurations in the neighbourhood of the current one. To obtain this behaviour we consider $g=2 \cdot c+1$ groups of ants (each of m ants), being c the number of adapted parameters. Then g is the number alternative configurations that we test to adjust the involved parameters. Each group of ants use the same values for the parameters. For each considered parameter π we fixed a probing variation $\Delta\pi$ and we test two alternative values from the current one, i.e., $\pi - \Delta\pi$ and $\pi + \Delta\pi$. In our experimental analysis, four alternative configurations are produced in the neighbourhood of the current values of q_0 and β , obtained keeping fixed q_0 and varying β of $\pm\Delta\beta$, and symmetrically keeping fixed β and varying q_0 of $\pm\Delta q_0$. We set both Δq_0 and $\Delta\beta$ equal to 0.1. After all the ants have constructed their solution, the local search is applied to the best solution found by each group: then, the current values of the parameters are changed to the ones that produced the best solution in the last iteration, considering also the local search effect. This adaptation mechanism is

activated at the start of the algorithm and stopped after 10 consecutive iterations without changes in the values of the parameters, since we consider the occurrence of this condition denoting that the adaptation found a good point of equilibrium of the parameter values. Finally, note that, whenever this adaptation mechanism is active, the algorithm does not perform the local pheromone update as this would introduce a cross influence in the exploration of the groups of the ants that we want to avoid during self-tuning.

4 Experimental Results

In order to evaluate the effectiveness of the proposed approaches, we coded ACO and AD-ACO algorithms in C++ and we executed an experimental campaign on an Intel Core2 CPU 6600 2.4 GHz, 2 Gb PC. We used the set of benchmark instances generated by V. Cicirello [7]¹. Such benchmark set is made of 120 instances with 60 jobs and it was produced by generating 10 instances for each combination of the three factors, the due date tightness δ , the due date range R , and the setup time severity ξ [16] varying in the following ranges: $\delta \in \{0.3, 0.6, 0.9\}$, $R \in \{0.25, 0.75\}$, $\xi \in \{0.25, 0.75\}$.

In our tests we used a set of $m=30$ ants (as in [13]) and we fixed as termination condition the maximum number of non improving iteration equal to 100. Then, in order to compare the behaviour of our basic ACO with the one of our AD-ACO, we first performed an extended preliminary tuning analysis experiment on a subset of instances to determine the suitable values for all the ACO parameters. In particular, we carried out a fully factorial design on the parameters using the following sets of values: $\alpha \in \{0.05, 0.1, 0.15, 0.2\}$, $\beta \in \{0.5, 1, 1.5, 2\}$, $\rho \in \{0.03, 0.05, 0.07, 0.09\}$ and $q_0 \in \{0.6, 0.7, 0.8, 0.9\}$. Such analysis did not highlight a clearly dominant parameter configuration among the tested ones; then we fixed the following values as the ones producing the smallest mean rank from the well-known non parametric Friedman's statistical test: $\alpha=0.1$, $\beta=0.5$, $\rho=0.03$ and $q_0=0.7$. We must remark that the upper bound τ_{max} was arbitrary fixed to 100 within the ACO code, so determining an initial pheromone value $\tau_0=50$: then both such values are not algorithm parameters.

We first compared the average results produced by ACO and AD-ACO algorithms. We tested two starting configurations for parameters β and q_0 which are dynamically adapted in AD-ACO: configuration (a) started with the same basic ACO values $\beta=0.5$ and $q_0=0.7$, whereas configuration (b) with $\beta=3$ and $q_0=0.5$. For each algorithm we executed 10 runs on the benchmark set and then we computed the best average result for each instance b_{avg} . We evaluated for each instance the average percentage deviation as $100 \cdot ((avg - b_{avg}) / b_{avg})$, with both avg and b_{avg} greater than zero (we set zero deviation when both the values are zero; otherwise we discarded the sample if only one value is zero). Table 1 reports the comparison among the results: the first pair of columns shows the overall average percentage deviations from the best averages (*Avg dev*) and the relevant standard deviation (*Stdev*), the *Mean Rank* column gives the

¹ The benchmark used in this paper is available online at <http://www.cs.drexel.edu/~cicirello/benchmarks.html>

Table 1. The comparison of the overall average ACO and AD-ACO results

	Avg dev	Stdev	Mean Rank	Avg CPU	without outliers	
					Avg dev	Stdev
AD-ACO _b	8.04%	64.08%	1.88	48.29	0.27%	1.28%
AD-ACO _a	1.09%	3.07%	2.79	21.84	0.60%	0.90%
ACO	5.12%	38.44%	3.87	12.24	1.06%	1.44%

output of the non-parametric Friedman’s test that confirmed the statistical relevance of the differences in the results at 5% significance; *Avg CPU* provides the average CPU time needed; the last pair of columns reports the overall average results excluding outliers (we excluded the instances with a percentage deviation not in the interval [-40%, 40%]): in fact, since in the objective values in the benchmark we observed differences of several orders of magnitude, the elimination of the outliers would reduce the possible influence of very slight absolute differences in the objectives for instances with small reference values.

From Table 1 we can conclude that AD-ACO outperformed our basic ACO: in particular, just the algorithm starting from configuration (b), which was not obtained by the preliminary tuning, produced the best average results. This is remarked both by the statistical test and by the average results excluding the outliers that allow us to conclude that AD-ACO dominates the non-adaptative algorithm, so confirming the effectiveness of the proposed adaptation mechanism.

Even if we believe that the effectiveness of a stochastic algorithm must be evaluated from its average results, we considered also the best results obtained over 10 runs. We compared our best solutions with the best known results (hereinafter denoted as BKR) so far published in literature for the same benchmark, which are mainly produced by one among the SA, TS and GA algorithms presented by Lin and Ying [12] (102 best known results over 120), plus 15 best known results due to the ACO defined by Liao and Juan [13] and 3 to the VBSS algorithm introduced by Cicerello [10]. Note that all the mentioned algorithms found the 16 zero-cost instances so far discovered. In addition, we compared the ACO and AD-ACO results with the set of best known solutions updated taking into account the solutions yielded by a new discrete particle swarm optimization (DPSO) algorithm that we recently presented [17] (we denoted this set of best known solutions as BKR+DPSO). The relevant comparisons can be seen in Table 2. Also from Table 2 the dominance of our AD-ACO over the basic ACO clearly emerges. Both ACO and AD-ACO produced a relevant number of better results with respect to the BKR ones, with a significant percent average improvement. However, we must also note that our DPSO approach seems, at least when considering the produced best known solution, still superior. A further investigation and comparison of all our approaches will be the subject of a future experimental campaign.

Finally, we can notice the typical trend of the parameter β during the iterations of the algorithm, which tends to a value close to zero iteration after iteration. This is an expected behaviour since in the first phases of the algorithm the pheromone values are not very dissimilar, so the heuristic information is necessary to help ants finding good solutions. After that ants have learned which are the “good” solutions components, the heuristic information is no more necessary; it is indeed harmful, because it does not allow ants to diversify solutions from the one suggested by the heuristic.

Table 2. The comparison of the ACO and AD-ACO best results with the BKR and BKR+DPSO best known solutions

	Comparison with BKR				Comparison with BKR+ DPSO			
	% Dev				% Dev			
		without outliers	impr bks	% impr bks		without outliers	impr bks	% impr bks
AD-ACO _b	-2,34%	-1,91%	88	73,33%	4,47%	0,32%	26	21,67%
AD-ACO _a	-2,73%	-1,96%	86	71,67%	2,29%	0,65%	26	21,67%
ACO	-2,23%	-1,40%	80	66,67%	2,40%	0,98%	12	10,00%

5 Conclusions

In this paper we proposed a new AD-ACO algorithm including a parameter adaptation mechanism that tries to reduce the need of parameter tuning. This mechanism, starting from an initial set of parameter values, aims at finding at runtime the best configuration for a subset of parameters with a simple neighbourhood search procedure. In particular, we tested this method on parameters q_0 and β . The adaptive algorithm is based on an ACO approach that we first recently proposed [6], whose most relevant features have been highlighted in this paper. In our ACO we do not need to define a quality function to perform the pheromone update and consequently we can avoid the choice of the initial pheromone value, which typically is a thorny problem. In addition, the adaptation mechanism makes the choice of the starting values of parameters q_0 and β less critical. Therefore, the remaining parameters to be set are m , the number of ants, β , which plays a fundamental role since it strongly affects the convergence rate, and ρ used for intra-iteration diversification. Given that we generally observed a low sensitivity of our ACO from these parameters, we must admit that we did not find any clear rationale in changing them at runtime.

We tested the proposed approach on the single machine total tardiness scheduling problem with sequence-dependent setups. The comparison test on a well-known benchmark pointed out the quality of AD-ACO and the ability of both the algorithms to improve the benchmark best known results provided by the previous approaches of reference.

References

1. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge (2004)
2. Dorigo, M.: Optimization, learning and natural algorithms (in Italian). PhD Thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy (1992)
3. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
4. Kennedy, J., Eberhart, R.C.: Swarm Intelligence. Morgan Kaufmann Publishers, San Francisco (2001)
5. Stützle, T., Hoos, H.H.: Max-min ant system. Future Generation Computer System 16, 889–914 (2000)

6. Anghinolfi, D., Paolucci, M.: A new ant colony optimization approach for the single machine total weighted tardiness scheduling problem. *International Journal of Operations Research* 5(1), 1–17 (2008)
7. Cicirello, V.A.: Weighted tardiness scheduling with sequence-dependent setups: a benchmark library. Technical Report. Intelligent Coordination and Logistics Laboratory. Robotics Institute. Carnegie Mellon University, USA (2003)
8. Lawler, E.L.: A ‘pseudopolynomial’ algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics* 1, 331–342 (1997)
9. Lee, Y.H., Bhaskaran, K., Pinedo, M.: A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transaction* 29, 45–52 (1997)
10. Cicirello, V.A., Smith, S.F.: Enhancing stochastic search performance by value-based randomization of heuristics. *Journal of Heuristics* 11, 5–34 (2005)
11. Cicirello, V.A.: Non-Wrapping Order Crossover: An Order Preserving Crossover Operator that Respects Absolute Position. In: *Proceedings of GECCO 2006 Conference*, Seattle, WA, USA, pp. 1125–1131 (2006)
12. Lin, S.-W., Ying, K.-C.: Solving single-machine total weighted tardiness problems with sequence-dependent setup times by meta-heuristics. *The International Journal of Advanced Manufacturing Technology* (2006)
13. Liao, C.-J., Juan, H.C.: An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. *Computers & Operations Research* 34, 1899–1909 (2007)
14. Dorigo, M., Gambardella, L.M.: Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation* 1, 53–66 (1997)
15. Tasgetiren, M.F., Sevkli, M., Liang, Y.-C., Gencyilmaz, G.: Particle swarm optimization algorithm for single machine total weighted tardiness problem. In: *Proceedings of the 2004 Congress on Evolutionary Computation (CEC 2004)*, Portland, Oregon, pp. 1412–1419 (2004)
16. Pinedo, M.: *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, Englewood Cliffs (1995)
17. Anghinolfi, D., Paolucci, M.: A new discrete particle swarm optimization approach for the total tardiness scheduling problem. *European Journal of Operational Research* (2007)

Robust Optimization by ϵ -Ranking on High Dimensional Objective Spaces

Hernán Aguirre^{1,2} and Kiyoshi Tanaka²

¹ Fiber-Nanotech Young Researcher Empowerment Program

² Shinshu University, Faculty of Engineering,
4-17-1 Wakasato, Nagano, 380-8553 Japan
{ahernan,ktanaka}@shinshu-u.ac.jp

Abstract. This work proposes a method to fine grain the ranking of solutions after they have been ranked by Pareto dominance, aiming to improve the performance of evolutionary algorithms on *many* objectives optimization problems. The re-ranking method uses a randomized sampling procedure to choose, from sets of equally ranked solutions, those solutions that will be given selective advantage. The sampling procedure favors a good distribution of the sampled solutions based on dominance regions wider than conventional Pareto dominance. We enhance NSGA-II with the proposed method and test its performance on MNK-Landscapes with up to $M = 10$ objectives. Experimental results show that convergence and diversity of the solutions found can improve remarkably on $3 \leq M \leq 10$ objectives problems.

1 Introduction

Multiobjective evolutionary algorithms (MOEAs) [1,2] optimize simultaneously two or more objective functions, aiming to find a set of compromised Pareto optimal solutions in a single run of the algorithm. Most state of the art MOEAs use Pareto dominance within the selection procedure of the algorithm to rank solutions. Selection based on Pareto dominance is thought to be effective for problems with convex and non-convex fronts and has been successfully applied in two and three objectives problems.

Recently, there is a growing interest on applying MOEAs to solve *many* objectives optimization problems, i.e. problems with four or more objectives. However, current research reveals that the number of Pareto non-dominated solutions increase substantially as we increase the number of objectives of the problem [3,4]. Hence, ranking by Pareto dominance becomes coarser and many solutions are assigned the same rank. This affects the effectiveness of selection, severely deteriorating the performance of MOEAs [5,6,7].

In this work, we propose a method to enhance selection of MOEAs aiming to improve their performance on *many* objectives optimization problems. The proposed method fine grains the ranking of solutions after they have been ranked by Pareto dominance. The re-ranking method uses a randomized sampling procedure to choose, from sets of equally ranked solutions, those solutions that will

be given selective advantage. The sampling procedure favors a good distribution based on dominance regions wider than conventional Pareto dominance. Thus, the proposed ranking method increases selection probabilities of some of the solutions, while trying to keep a uniform search effort towards the different zones of objective space represented in the actual population.

In this paper, we enhance NSGA-II [8] with the proposed method. NSGA-II is a widely referenced algorithm that performs relatively well on multiobjective problems, but known to perform poorly on *many* objective problems. We test the enhanced algorithm on MNK-Landscapes [3,4] with $2 \leq M \leq 10$ objectives and $0 \leq K \leq 50$ epistatic interactions, varying the parameter that determines the domination region of the sampled solutions. Experimental results show that convergence and diversity of the solutions found can improve remarkably on $3 \leq M \leq 10$ objectives for most K .

2 Multiobjective Optimization Concepts and Definitions

Let us consider, without loss of generality, a maximization multiobjective problem with M objectives:

$$\text{maximize } \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_M(\mathbf{x})) \quad (1)$$

where $\mathbf{x} \in \mathcal{S}$ is a solution vector in the feasible solution space $\mathcal{S}$, and $f_1(\cdot), f_2(\cdot), \dots, f_M(\cdot)$ the M objectives to be maximized. Two concepts used in multiobjective optimization that are of special relevance to this work are as follows.

Definition 1 (Pareto dominance). *A solution $\mathbf{x}$ is said to Pareto dominate other solution $\mathbf{y}$ if the two following conditions are satisfied:*

$$\begin{aligned} \forall m \in \{1, \dots, M\} \quad f_m(\mathbf{x}) &\geq f_m(\mathbf{y}) \quad \wedge \\ \exists m \in \{1, \dots, M\} \quad f_m(\mathbf{x}) &> f_m(\mathbf{y}). \end{aligned} \quad (2)$$

Here, $\mathbf{x}$ dominates $\mathbf{y}$ is denoted by $\mathbf{f}(\mathbf{x}) \succeq \mathbf{f}(\mathbf{y})$.

In decision making, one dimensional comparison and Pareto optimality are two popular methods used to choose a solution once a set of Pareto optimal solutions has been found. Yu [9] showed that these two methods are two extreme cases in the entire domain of domination structures and that there are infinity valid methods lying between them, which suitability depends on how much information is known on the decision maker's preferences. Within the EMO community, these other domination structures are also known as relaxed forms of Pareto dominance and one method to implement them is ϵ -dominance [10].

Definition 2 (ϵ -dominance). *A solution $\mathbf{x}$ is said to ϵ -dominate other solution $\mathbf{y}$ if the two following conditions are satisfied:*

$$\begin{aligned} \forall m \in \{1, \dots, M\} \quad (1 + \epsilon)f_m(\mathbf{x}) &\geq f_m(\mathbf{y}) \quad \wedge \\ \exists m \in \{1, \dots, M\} \quad (1 + \epsilon)f_m(\mathbf{x}) &> f_m(\mathbf{y}). \end{aligned} \quad (3)$$

where $\epsilon > 0.0$. Here, $\mathbf{x}$ ϵ -dominates $\mathbf{y}$ is denoted by $\mathbf{f}(\mathbf{x}) \succeq^\epsilon \mathbf{f}(\mathbf{y})$.

Fig. 1 illustrates for one solution its Pareto dominance region and its ϵ -dominance region.

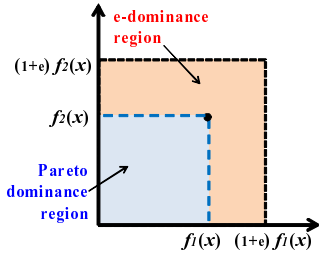


Fig. 1. Pareto dominance and ϵ -dominance

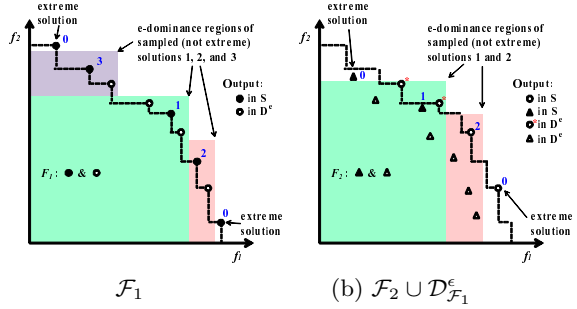


Fig. 2. ϵ -sampling on (a) the first front and (b) second front joined with solutions demoted from the first

3 Method

In this section, we describe the proposed method to fine grain the ranking of solutions initially ranked by Pareto dominance. The core of the proposed method is a randomized sampling procedure that favors a good distribution of solutions based on dominance regions wider than conventional Pareto dominance. In the following, we first explain the sampling procedure and then a re-ranking procedure for NSGA-II [8], the algorithm used in our experiments.

3.1 ϵ -Sampling

ϵ -sampling assumes that there is a set of equally ranked solutions from which a subset of them should be chosen to be given selective advantage in order to proceed further with the evolutionary search. That is, ϵ -sampling acts as a decision making procedure, not to find a final solution, but to help selection of the evolutionary algorithm. Hence, the sampling heuristic must reflect criteria that favor an effective search. Here, the sample of solutions to be given selective advantage are obtained with the following criteria,

- Extreme solutions are always part of the sample.
- Each (not extreme) sampled solution is the sole sampled representative of its area of influence. The area of influence of the sampled solutions is determined by a domination region wider than Pareto dominance, i.e. ϵ -dominance.
- Sampling of (not extreme) solutions follows a random schedule.

The first criterion tries to push the search towards the optimum values of each fitness function, aiming to find non-dominated solutions in a wide area of objective space. The second criterion assures that only one solution in a given zone of objective space is given higher rank, trying to distribute the search effort more or less uniformly among the different zones represented in the actual population. The third criterion dynamically establishes the zones that are represented in the sample. Also, in the case that there are several solutions within each zone,

it increases the likelihood that the sampled solutions that will be given higher rank are different from one generation to the next, increasing the possibility of exploring wider areas of objective and variable space. This is an important issue in elitist algorithms, like most state of the art MOEAs. Summarizing, the proposed sampling method is a decision making procedure used by the algorithm in order to increase selection probabilities of some of the solutions, trying to keep a uniform search effort towards the different zones of objective space.

Procedure 1 illustrates the algorithm of the proposed ϵ -sampling method. Let us denote $\mathcal{A}$ the set of solutions that have been assigned the same rank based on conventional Pareto dominance, for example by applying non-domination sorting [8]. ϵ -sampling returns the sampled solutions $\mathcal{S} \subset \mathcal{A}$ that will be given selective advantage as well as the set of solutions $\mathcal{D}^\epsilon$ to be demoted. See that extreme solutions are the first to be assigned to the sample $\mathcal{S}$ (lines 1,2). Then, one by one, solutions are randomly chosen and included in $\mathcal{S}$ (lines 6-8), whereas solutions that lie in the wider domination region of the randomly picked solution are assigned to $\mathcal{D}^\epsilon$ (lines 9,10).

Procedure 1. ϵ -sampling (ϵ , $\mathcal{A}$, $\mathcal{S}$, $\mathcal{D}^\epsilon$)

Input: ϵ -dominance factor ϵ and a set of solutions $\mathcal{A}$

Output: $\mathcal{S}$ and $\mathcal{D}^\epsilon$ ($\mathcal{S} \cup \mathcal{D}^\epsilon = \mathcal{A}$). $\mathcal{S}$ contains extreme and ϵ -non-dominated solutions, whereas $\mathcal{D}^\epsilon$ contains ϵ -dominated solutions

```

1:  $\mathcal{X} \leftarrow \{x \in \mathcal{A} \mid f_m(x) = \max(f_m(\cdot)), m = 1, 2, \dots, M\}$ 
2:  $\mathcal{S} \leftarrow \mathcal{X}$ 
3:  $\mathcal{A} \leftarrow \mathcal{A} \setminus \mathcal{X}$ 
4:  $\mathcal{D}^\epsilon \leftarrow \emptyset$ 
5: while  $\mathcal{A} \neq \emptyset$  do
6:    $r \leftarrow \text{rand}(), 1 \leq r \leq |\mathcal{A}|$ 
7:    $z \leftarrow r\text{-th solution} \in \mathcal{A}$ 
8:    $\mathcal{S} \leftarrow \mathcal{S} \cup \{z\}$ 
9:    $\mathcal{Y} \leftarrow \{y \in \mathcal{A} \mid z \succeq^\epsilon y, z \neq y\}$ 
10:   $\mathcal{D}^\epsilon \leftarrow \mathcal{D}^\epsilon \cup \mathcal{Y}$ 
11:   $\mathcal{A} \leftarrow \mathcal{A} \setminus \{\{z\} \cup \mathcal{Y}\}$ 
12: end while
13: return
```

Procedure 2. ϵ -ranking (ϵ , $\mathcal{F}$, $\mathcal{F}^\epsilon$)

Input: ϵ -dominance factor ϵ and solutions $\mathcal{F}$ classified in fronts $\mathcal{F}_i$ ($i = 1, \dots, N_F$) by non-domination sorting

Output: $\mathcal{F}^\epsilon$, solutions re-classified in fronts $\mathcal{F}_j^\epsilon$ ($j = 1, \dots, N_F^\epsilon$) after ϵ -sampling

```

1:  $\mathcal{D}^\epsilon \leftarrow \emptyset, i \leftarrow 1, j \leftarrow 1$ 
2: repeat
3:   if  $i \leq N_F$  then
4:      $\mathcal{A} \leftarrow \mathcal{F}_i \cup \mathcal{D}^\epsilon$ 
5:      $i \leftarrow i + 1$ 
6:   else
7:      $\mathcal{A} \leftarrow \mathcal{D}^\epsilon$ 
8:   end if
9:    $\epsilon\text{-sampling}(\epsilon, \mathcal{A}, \mathcal{S}, \mathcal{D}^\epsilon)$ 
10:   $\mathcal{F}_j^\epsilon \leftarrow \mathcal{S}$ 
11:   $j \leftarrow j + 1$ 
12: until  $\mathcal{D}^\epsilon = \emptyset$ 
13: return
```

3.2 ϵ -Ranking in NSGA-II

The ϵ -sampling procedure works on a set of equally ranked solutions, however within a population there could be several sets of such solutions (each set with a different rank). Here, we explain ϵ -ranking for NSGA-II to re-rank all possible sets of equally ranked solutions using the ϵ -sampling method.

In NSGA-II, a non-domination sorting procedure is applied to the joined population of parents and offspring to classify solutions in fronts of non-dominated solutions. Let us denote $\mathcal{Z}$ the set of solution we want to classify. The first front $\mathcal{F}_1$ is obtained from $\mathcal{Z}$ and corresponds to the set of Pareto optimal solutions in $\mathcal{Z}$. Let us denote this set as $\mathcal{POS}_1$. The subsequent fronts $\mathcal{F}_i; i > 1$, contain lower level non-dominated solutions and are obtained by disregarding solutions corresponding to the previously classified fronts, i.e. $\mathcal{F}_i; i > 1$, is obtained from the set $\mathcal{Z} - \bigcup_{k=1}^{i-1} \mathcal{POS}_k$. Solutions in each front are assigned the same non-domination rank, equal to the front number they belong to.

ϵ -ranking in NSGA-II is applied at each generation after non-domination sorting to reclassify the fronts $\mathcal{F}_i$ ($i = 1, \dots, N_F$) found by NSGA-II. **Procedure 2** describes the ϵ -ranking method for NSGA-II. See that the reclassified front $\mathcal{F}_j^\epsilon$ ($j = 1, \dots, N_F^\epsilon$) now contains only the sample of solutions $\mathcal{S} \subset \mathcal{F}_i$ found by ϵ -sampling (lines 9,10). Also, see that solutions $\mathcal{D}^\epsilon$, which are not part of the sample (line 9) are demoted by joining them with solutions of an inferior front in the next iteration of the loop (line 4). Thus, $\mathcal{F}_1^\epsilon$ contains some of the solutions initially ranked first, but $\mathcal{F}_j^\epsilon, j > 1$, can contain solutions that initially were ranked in different fronts. This gives chance to lateral diversity present in the initial ranking of solutions and can punish highly crowded solutions even if they are initially ranked first by conventional Pareto dominance. **Fig. 2** illustrates the application of ϵ -sampling on the first front $\mathcal{F}_1$ and on the second front $\mathcal{F}_2$ joined with the demoted solutions $\mathcal{D}^\epsilon$ from $\mathcal{F}_1$. A number close to the solutions represents the random schedule in which solutions are sampled (0 means extreme solutions, which are all selected at once).

Truncation to obtain the new parent population is applied after re-classifying fronts with ϵ -ranking. Many demoted solutions would still be part of the new parent population, though their probability to reproduce has been reduced instantaneously by assigning them an inferior rank. Demoted solutions, which are not dominated by offspring, could be part of the highest ranked sample in the next generation, because ϵ -sampling would be applied again in a randomized manner.

4 Test Problems, Performance Measures and Parameters

4.1 Multiobjective MNK-Landscapes

In this work we test the performance of the algorithms on multiobjective MNK-Landscapes. A multiobjective MNK-Landscape [3,4] is defined as a vector function mapping binary strings into real numbers $\mathbf{f}(\cdot) = (f_1(\cdot), f_2(\cdot), \dots, f_M(\cdot)) : \mathcal{B}^N \rightarrow \mathbb{R}^M$, where M is the number of objectives, $f_i(\cdot)$ is the i -th objective function, $\mathcal{B} = \{0, 1\}$, and N is the bit string length. $\mathbf{K} = \{K_1, \dots, K_M\}$ is a set of integers where K_i ($i = 1, 2, \dots, M$) is the number of bits in the string that epistatically interact with each bit in the i -th landscape. Each $f_i(\cdot)$ can be expressed as an average of N functions as follows

$$f_i(\mathbf{x}) = \frac{1}{N} \sum_{j=1}^N f_{i,j}(x_j, z_1^{(i,j)}, z_2^{(i,j)}, \dots, z_{K_i}^{(i,j)}) \quad (4)$$

where $f_{i,j} : \mathcal{B}^{K_i+1} \rightarrow \mathbb{R}$ gives the fitness contribution of bit x_j to $f_i(\cdot)$, and $z_1^{(i,j)}, z_2^{(i,j)}, \dots, z_{K_i}^{(i,j)}$ are the K_i bits interacting with bit x_j in the string $\mathbf{x}$. The fitness contribution $f_{i,j}$ of bit x_j is a number between $[0.0, 1.0]$ drawn from a uniform distribution. Thus, each $f_i(\cdot)$ is a non-linear function of $\mathbf{x}$ expressed by a Kauffman's NK-Landscape model of epistatic interactions [11]. In addition, it is also possible to arrange the epistatic pattern between bit x_j and the K_i other interacting bits. That is, the distribution $D_i = \{\text{random}, \text{nearest neighbor}\}$ of K_i bits among N . Thus, $M, N, \mathbf{K} = \{K_1, K_2, \dots, K_M\}$, and $\mathbf{D} = \{D_1, D_2, \dots, D_M\}$, completely specify a multiobjective MNK-Landscape.

4.2 Performance Measures

In this work, we use the hypervolume $\mathcal{H}$ and coverage $\mathcal{C}$ measures [12] to evaluate and compare the performance of the algorithms. The measure $\mathcal{H}$ calculates the volume of the M -dimensional region in objective space enclosed by a set of non-dominated solutions and a dominated reference point. Let $\mathcal{A}$ be a set of non-dominated solutions. The hypervolume of $\mathcal{A}$ can be expressed as

$$\mathcal{H}(\mathcal{A}) = \bigcup_{i=1}^{|\mathcal{A}|} (\mathcal{V}_i - \bigcap_{j=1}^{i-1} \mathcal{V}_i \mathcal{V}_j) \quad (5)$$

where $\mathcal{V}_i$ is the hypervolume rendered by the point $\mathbf{x}_i \in \mathcal{A}$ and the reference point. In this work, the reference point is set to $[0.0, \dots, 0.0]$. Given two sets of non-dominated solutions $\mathcal{A}$ and $\mathcal{B}$, if $\mathcal{H}(\mathcal{A}) > \mathcal{H}(\mathcal{B})$ then set $\mathcal{A}$ can be considered better on convergence and/or diversity of solutions. To calculate $\mathcal{H}$, we use Fonseca et al. [13] algorithm, which significantly reduces computational time.

The coverage $\mathcal{C}$ measure [12] provides complementary information on convergence. Let us denote $\mathcal{A}$ and $\mathcal{B}$ the sets of non-dominated solutions found by two algorithms. $\mathcal{C}(\mathcal{A}, \mathcal{B})$ gives the fraction of solutions in $\mathcal{B}$ that are dominated at least by one solution in $\mathcal{A}$. More formally,

$$\mathcal{C}(\mathcal{A}, \mathcal{B}) = \frac{|\{\mathbf{b} \in \mathcal{B} | \exists \mathbf{a} \in \mathcal{A} : \mathbf{f}(\mathbf{a}) \succeq \mathbf{f}(\mathbf{b})\}|}{|\mathcal{B}|}. \quad (6)$$

$\mathcal{C}(\mathcal{A}, \mathcal{B}) = 1.0$ indicates that all solutions in $\mathcal{B}$ are dominated by solutions in $\mathcal{A}$, whereas $\mathcal{C}(\mathcal{A}, \mathcal{B}) = 0.0$ indicates that no solution in $\mathcal{B}$ is dominated by solutions in $\mathcal{A}$. Since usually $\mathcal{C}(\mathcal{A}, \mathcal{B}) + \mathcal{C}(\mathcal{B}, \mathcal{A}) \neq 1.0$, both $\mathcal{C}(\mathcal{A}, \mathcal{B})$ and $\mathcal{C}(\mathcal{B}, \mathcal{A})$ are required to understand the degree to which solutions of one set dominate solutions of the other set.

4.3 Parameters

In this work, we test the performance of the algorithm using MNK-Landscapes with $2 \leq M \leq 10$ objectives, $N = 100$ bits, number of epistatic interactions $K = \{0, 1, 3, 5, 10, 15, 25, 35, 50\}$ ($K_1, \dots, K_M = K$), and *random* epistatic patterns among bits for all objectives ($D_1, \dots, D_M = \text{random}$). Results presented below show the average performance of the algorithms on 50 different problems

randomly generated for each combination of M , N and K . In the plots, error bars show 95% confidence intervals on the mean.

In the following sections we analyze results by NSGA-II enhanced with the proposed method (referred as ϵ -ranking for short) comparing them with results by conventional NSGA-II. ϵ -ranking and conventional NSGA-II use parent and offspring populations of size 100, two point crossover for recombination with rate $p_c = 0.6$, and bit flipping mutation with rate $p_m = 1/N$ per bit. The number of evaluations is set to 3×10^5 . To have a broad picture of the effect of the wider dominations regions used within ϵ -sampling, in our study we conduct experiments varying ϵ on the range $[0.5, 10.0]$ (%) in intervals of 0.5 for $2 \leq M \leq 6$ objectives problems; whereas for $7 \leq M \leq 10$ objectives we vary ϵ on the ranges $[1.0, 10.0]$ (%) in intervals of 0.5 and $[12.5, 22.5]$ (%) in intervals of 2.5.

5 Experimental Results and Discussion

In this section, we first discuss the relative gains on performance by ϵ -ranking set with ϵ^* that achieves maximum hypervolume $\mathcal{H}$. **Fig. 3** shows the average ratio $\frac{\mathcal{H}(E)}{\mathcal{H}(N)}$, where E and N denote the set of solutions found by ϵ -ranking and conventional NSGA-II, respectively. Thus, a ratio greater than 1.0 indicates better $\mathcal{H}$ by ϵ -ranking than conventional NSGA-II. As a reference, we include a horizontal line to represent the $\mathcal{H}(N)$ values normalized to 1.0. From this figure, we can see that ϵ -ranking can slightly improve $\mathcal{H}$ on problems with $M = 2$ and $M = 3$ objectives for some values of K (4% improvement or less). On the other hand, for $4 \leq M \leq 10$ objectives, the improvement on $\mathcal{H}$ is remarkable for most values of K (up to 27% improvement). Note that improvements on $\mathcal{H}$ become larger as we increase the number of objectives M from 2 to 6, whereas improvements on $\mathcal{H}$ are similar for $7 \leq M \leq 10$.

Improvements on $\mathcal{H}$ can be due to solutions with better convergence, better diversity, or both. To complement the analyzes of results on $\mathcal{H}$ we also present results using the $\mathcal{C}$ measure. **Fig. 4** shows the average $\mathcal{C}$ values between conventional NSGA-II and ϵ -ranking set with ϵ^* . From this figure, we can be see that $\mathcal{C}(E, N)$ is slightly smaller than $\mathcal{C}(N, E)$ for $M = 2$ and $K \leq 10$, which means that convergence is somewhat worse by ϵ -ranking than conventional NSGA-II. Thus, the slight improvement on $\mathcal{H}$ by ϵ -ranking, observed in the same region $M = 2$ and $0 \leq K \leq 10$ as shown in **Fig. 3**, is at the expense of a slight deterioration on convergence.

In the case of $M = 3$ objectives, we can see that $\mathcal{C}(E, N)$ is considerably greater than $\mathcal{C}(N, E)$ for $0 \leq K \leq 25$, which means that convergence is better by ϵ -ranking than conventional NSGA-II. Thus, we can conclude that a better convergence of solutions contributes to the improvement on $\mathcal{H}$ by ϵ -ranking, as shown in **Fig. 3** for $0 \leq K \leq 25$.

For $4 \leq M \leq 10$ a clear trend can be seen. $\mathcal{C}(N, E)$ is close to 0.0 for most K and M . This indicates that there are almost no solutions by conventional NSGA-II that dominate solutions by ϵ -ranking. On the other hand, the values of $\mathcal{C}(E, N)$ are very high for 4 objectives (in the range 0.55-0.85) and reduce

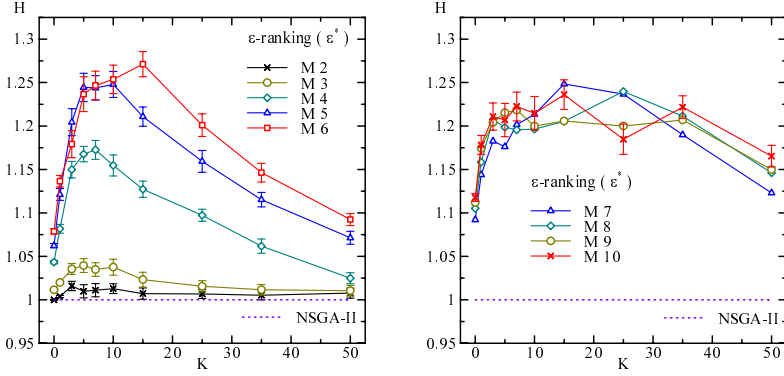


Fig. 3. Normalized $\mathcal{H}$, ϵ -ranking set with ϵ^* that achieves maximum $\mathcal{H}(E)$

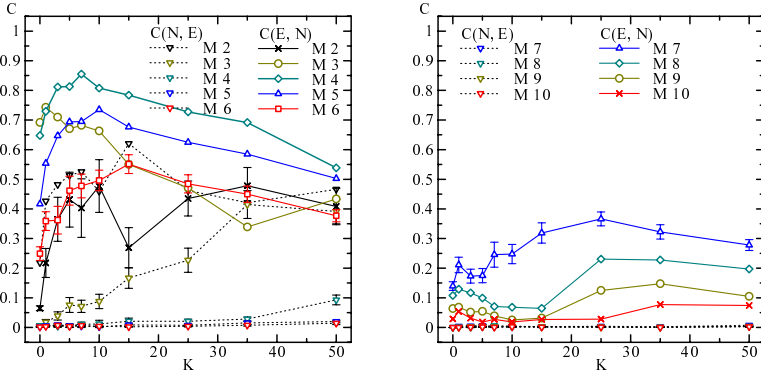


Fig. 4. $\mathcal{C}$ between NSGA-II and ϵ -ranking set with ϵ^* that achieves maximum $\mathcal{H}(E)$

gradually as we increase M up to 10 objectives (in the range 0.01-0.08). This suggests that a better convergence of solutions contributes to the increases of $\mathcal{H}$ by ϵ -ranking on $M = 4$ problems. As we increase M , gains on diversity gradually become more significant than gains on convergence as the reason for the remarkable improvement of $\mathcal{H}$ on $5 \leq M \leq 10$.

Next, we briefly discuss the effects on performance of parameter ϵ . **Fig. 5** and **Fig. 6** show for $M = 6$ objectives the ratio $\frac{\mathcal{H}(\epsilon)}{\mathcal{H}(N)}$ and the $\mathcal{C}$ measure between NSGA-II and ϵ -ranking varying ϵ . From **Fig. 5**, we can see that $\mathcal{H}$ improves as we increase ϵ , eventually reaching maximum $\mathcal{H}^*$ at ϵ^* . Increasing ϵ above ϵ^* has the effect to gradually reduce $\mathcal{H}$, specially on $1 \leq K \leq 25$ landscapes (skewed bell-like curves with larger tail for $\epsilon > \epsilon^*$). However, $\epsilon > \epsilon^*$ on $K = \{0, 35, 50\}$ seems not to affect $\mathcal{H}$ (ramp-like curves). Thus, the region where ϵ^* lies in parameter space could be narrow or broad and depends on both K and M (results are only presented for $M = 6$ due to space limitations). Note that for most bell-like curves, ϵ^* for different K (or an ϵ that leads to high $\mathcal{H}$) are clustered at close range. For example, ϵ^* for most K are in the range $[2.5, 4.5]$ on $M = 6$ objectives.

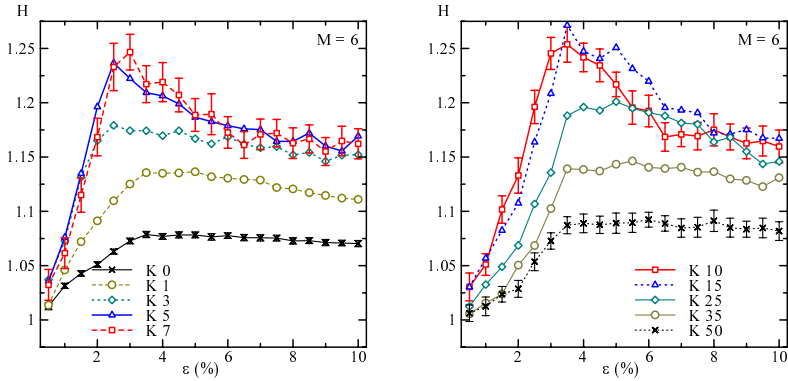


Fig. 5. $\mathcal{H}$ (normalized) varying ϵ , $M = 6$ objectives

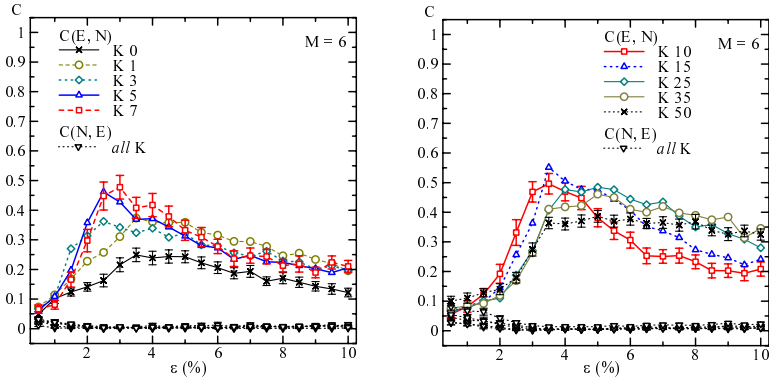


Fig. 6. $\mathcal{C}$ between NSGA-II and ϵ -ranking varying ϵ , $M = 6$ objectives

Looking at $\mathcal{C}$ plots in **Fig. 6**, it is interesting to notice that most $\mathcal{C}(E, N)$ curves are bell-like. This suggests that, although there could be a broad range of values for parameter ϵ that leads to high $\mathcal{H}$, the choice of ϵ would allow to focus on higher convergence or higher diversity compared to NSGA-II. Note that $\mathcal{C}(N, E)$ are close to 0.0 for most K (specially for large M). So, the choice of ϵ would not worsen convergence compared to NSGA-II (an exception could be $M = 2$, $K \leq 1$ and $M = 3$ $K = 0$ if ϵ is too large). Also, it is important to note that the value of ϵ that maximizes $\mathcal{H}$ is similar to the value of ϵ that maximizes $\mathcal{C}(E, N)$ for most K (and M). The patterns we observe here are useful to understand better the optimization of problems with *many* objectives. However, it would be interesting in the future to look for ways to set ϵ adaptively.

6 Conclusions

In this work, we have proposed a method to fine grain the ranking of solutions in Pareto dominance MOEAs aiming to improve their performance on *many*

objectives problems. The re-ranking method uses a randomized sampling procedure to increase selection probabilities of some of the solutions, while trying to keep a uniform search effort towards the different zones of objective space represented in the instantaneous population. We enhanced NSGA-II with the proposed method and tested its performance on MNK-Landscapes with $2 \leq M \leq 10$ objectives, $N = 100$ bits and $0 \leq K \leq 50$ epistatic interactions. We showed that both convergence and diversity of the obtained solutions can improve remarkably on problems with $3 \leq M \leq 10$ objectives for any level of epistatic interactions.

As future works, we would like pursue adaptive methods to control the dominance region used by the sampled solutions. Also, we would like to compare the proposed method with other approaches for *many* objectives optimization.

Acknowledgment

This study was performed through Special Coordination Funds for Promoting Science and Technology of the Ministry of Education, Culture, Sports, Science and Technology, of the Japanese Government.

References

1. Deb, K.: Multi-Objective Optimization using Evolutionary Algorithms. John Wiley & Sons, Chichester (2001)
2. Coello, C., Van Veldhuizen, D., Lamont, G.: Evolutionary Algorithms for Solving Multi-Objective Problems. Kluwer Academic Publishers, Boston (2002)
3. Aguirre, H., Tanaka, K.: Insights on Properties of Multiobjective MNK-Landscapes. In: Proc. 2004 IEEE Congress on Evolutionary Computation, pp. 196–203. IEEE Service Center (2004)
4. Aguirre, H., Tanaka, K.: Working Principles, Behavior, and Performance of MOEAs on MNK-Landscapes. European Journal of Operational Research 181(3), 1670–1690 (2007)
5. Purshouse, R.C., Fleming, P.J.: Conflict, Harmony, and Independence: Relationships in Evolutionary Multi-criterion Optimisation. In: Fonseca, C.M., Fleming, P.J., Zitzler, E., Deb, K., Thiele, L. (eds.) EMO 2003. LNCS, vol. 2632, pp. 16–30. Springer, Heidelberg (2003)
6. Aguirre, H., Tanaka, K.: Selection, Drift, Recombination, and Mutation in Multiobjective Evolutionary Algorithms on Scalable MNK-Landscapes. In: Coello Coello, C.A., Hernández Aguirre, A., Zitzler, E. (eds.) EMO 2005. LNCS, vol. 3410, pp. 355–369. Springer, Heidelberg (2005)
7. Hughes, E.J.: Evolutionary Many-Objective Optimisation: Many Once or One Many? In: Proc. 2005 IEEE Congress on Evolutionary Computation, vol. 1, pp. 222–227. IEEE Service Center (September 2005)
8. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II, KanGAL report 200001 (2000)
9. Yu, P.L.: Cone Convexity, Cone Extreme Points, and Nondominated Solutions in Decision Problems with Multiobjectives. Journal of Optimization Theory and Applications 14(3), 319–377 (1974)

10. Laumanns, M., Thiele, L., Deb, K., Zitzler, E.: Combining Convergence and Diversity in Evolutionary Multi-objective Optimization. *Evolutionary Computation* 10(3), 263–282 (Fall 2002)
11. Kauffman, S.A.: *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, Oxford (1993)
12. Zitzler, E.: *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*, PhD thesis, Swiss Federal Institute of Technology, Zurich (1999)
13. Fonseca, C., Paquete, L., López-Ibáñez, M.: An Improved Dimension-sweep Algorithm for the Hypervolume Indicator. In: *Proc. 2006 IEEE Congress on Evolutionary Computation*, pp. 1157–1163. IEEE Service Center (2006)

An Evolutionary Method for Natural Language to SQL Translation

Alexandre Afonso¹, Leonardo Brito¹, and Oto Vale²

¹ Escola de Engenharia Elétrica e de Computação-UFG
74605-010, Goiânia - Goiás - Brazil
brito@eeec.ufg.br

² Departamento de Letras, UFSCar
13.565-905, São Carlos - SP - Brazil
oto@letras.ufscar.br

Abstract. In this paper, we propose a new methodology where complex natural language requests from a user to a relational database are broken into simple sentences through an Evolutionary Computing method. Such basic sentences are then translated by another module, which tries to perform a pattern matching between a model filled by local grammars and the basic sentences generated by the Evolutionary Programming algorithm. The output of this system is a set of SQL queries to a specific database. The main feature is its combinatorial approach, as an alternative for the use of methods that employs many linguistic levels (lexicon, syntax rules and semantics) and intermediate languages. The proposed methodology is applied to Brazilian Portuguese. In our test bed, a 92% translation correctness was achieved.

Keywords: Evolutionary Computing, Natural Language Processing, Local Grammars, SQL.

1 Introduction

In this work, we propose the use of one of the paradigms [1] of Evolutionary Computing to solve a natural language translation problem: complex sentence partition into a set of basic atomic sentences.

After partition, the basic sentences become basic requests which, together, form a complete query. The basic sentence is defined as the minimal linguistic unit having SQL meaning. Local grammars [2][3] are the knowledge necessary to model natural language basic sentences among many linguistic levels at the same time (lexicon, syntax rules and semantics) by using only one formalism [4]. By using an Evolutionary Computing method to break complex requests into basic ones and local grammars to translate them, it is possible to create different language representations such as SQL. The chosen paradigm is Evolutionary Programming by applying a suitable mutation operator. The goal is to get a high degree of correct translations to simple and complex sentences in specific domains which have a particular vocabulary.

2 Natural Language Formalization

The first step when translating from natural language to SQL is to format the natural language sentence. The parts of a sentence are identified into the SQL context, then allowing the execution of the partition process. So, the original sentence must be identified syntactically and semantically as an SQL query by classes which we named *structural components* (Ps , Ta , R), as shown in the following example.

<i>Is There</i>	<i>a student</i>	<i>who has got a good grade?</i>
Existe	aluno	com nota alta ?
SELECT	*FROM Student	WHERE Student.Grade >= 7
Ps_I	Ta_I	R_I

We can define: Ps as a representation for the lexical items of selection, Ta as the database table name or synonym, and, R as a constraint (generally a phrase) on Ta or Ps . Some of these elements could be hidden, or they could be changed on its form, or they even could be in different positions through the sentence body, as

<i>Having a good grade</i>	<i>is there</i>	<i>any student?</i>
Com nota alta	existe	algum aluno?
R_I	Ps_I	Ta_I

<i>Students</i>	<i>having a good grade</i>	
Alunos	com nota alta	
Ta_I	R_I	

<i>Students</i>	<i>and teachers</i>	<i>having a good grade</i>
Alunos	e professores	com nota alta
Ta_I	Ta_2	R_I

The following *structural components sequences* are possible to basic sentences: $Ps_n Ta_m R_I!$, $Ps_n Ta_m!$, $Ps_n R_I!$, and R_I . The symbol ! means that the sequence could be a permutation. That is, $Ps_n Ta_m R_I$ is a possible structure, as much as the anagram $Ta_m R_I Ps_n$. If the symbol ! appears to the *structural components sequence* of a sentence, it means a *structural component sequence* was identified to the sentence but the system didn't identify the exact order of these *components*.

The constraint R_I is the main *structural component* identified inside a sentence, since it classifies a sentence as a basic one. Therefore, a basic sentence must have one constraint R_I and no more or less than one. This is the reason why R_I always has the subscribe I . If a sentence has more than one constraint R_I , it is a complex sentence, and so, it must be identified and broken into basic ones. However, the classes Ps and Ta can have more than one unit inside a basic sentence. The following example has three elements Ps and two elements Ta .

<i>Select, Count and make the average</i>	<i>to the students and teachers</i>	<i>having a good grade</i>
Selecione, conte e faça uma média	dos alunos e professores	com nota alta
Ps_I Ps_2 Ps_3	Ta_I Ta_2	R_I

We believe the number of units to Ps and Ta in a basic sentence is three at most. A Ps_n or Ta_m having more than three items ($n > 3$ or $m > 3$) is an atypical case.

2.1 Complex Sentences

The actual use of the language shows that requests might have a complex form rather than the simple sentence form shown before, although writing a complex sentence composed by more than three or four basic sentences is not a common use. Examples of complex sentences and their divisions into basic ones are:

1. (Is there a student having a good grade / or a bad grade?)
Existe aluno com nota alta / ou com nota baixa? – $Ps_1 Ta_1 R_1 / R_1$
2. (Having a good grade / or a reasonable grade, is there someone?)
Com nota alta / ou com nota razoável existe? – $R_1 / R_1 Ps_1$
3. (Is there a student or teacher having a good grade / or a employee having a bad grade?)
Existe aluno ou professor com nota alta / ou funcionário com nota baixa? – $Ps_1 Ta_1 Ta_2 R_1 / Ta_1 R_1$
4. (Is there a smart student / who has passed / but he is not living in Goiânia?)
Existe algum aluno inteligente / que foi aprovado / mas ele não mora em Goiânia? – $Ps_1 Ta_1 R_1 / Ta^* R_1 / Ta^* R_1$

The symbol “/” above shows where the complex sentences are broken to form simple sentences. Notice, the sentences formed by the partition are all simple ones and have exactly one constraint R_1 as shown by there *structural components sequences*. From the last example, the symbol * was used after a Ta symbol. It means that Ta is defined as a pronoun (in this case, *who* and *he*), and these pronouns were emitted by preview sentences (in this case, the first sentence) as a noun. Our goal is to propose an Evolutionary Algorithm to break up the complex sentences to simple ones by adding the symbol “/” at the correct place, and, additionally, this algorithm fills a stack that stores the nouns identified from each basic sentence. After the evolutionary process is done, the sentences that have a pronoun could use the information stored on the stack to solve anaphoric references by using an inferential processes.

(Is there a student having a good grade / who lives in Goiás / and it is greater than nine?) <i>Existe aluno com nota alta / que mora em Goiás / e ela é maior que nove?</i>	
Stack Two Subsequent Simple Sentences that Use the Nouns of the First Simple Sentence	
(grade) <i>nota</i>	(it is greater than nine) <i>ela é maior que nove</i>
(student) <i>aluno</i>	(who lives in Goiás) <i>que mora em Goiás</i>

Fig. 1. Anaphoric Resolution must be done after sentence partition by using the Stack of nouns

As shown in the previous example, part of anaphoric resolution is solved by the first step of the proposed methodology. That is, a sentence must be classified by *structural components sequences* and it is broken into simple sentences if more than a *constraint R_1* is identified. After that, a stack stores the nouns that appear among the identified basic sentences. Other basic sentences that have pronouns instead of nouns are analyzed by the second and final step: the translator module. This module is able

to translate each basic sentence to an SQL or a set of SQL queries. The translator, if necessary, tries to solve the anaphora problem by verifying the existence of a noun on the stack and by using an inferential process.

3 The Evolutionary Algorithm

The pseudo-code of the general Evolutionary Algorithm for breaking a complex sentence is as follows.

1. Load the auxiliary data structures for evaluation.
2. Load the input complex sentence and build its tags.
3. Generate the patriarch (the first individual).
4. Generate N individuals from the patriarch by mutation.
5. Evaluate the individuals, assigning a score to each one.
6. Select the best and reasonable individuals.
7. If stop criteria are satisfied (acceptable solution found, maximum elapsed search time or number of generations run), then Select the best solution found so far and Halt the procedure.
Else, Mutate K individuals having reasonable score by applying small perturbations to their chromosomes.
8. Select the best and reasonable individuals after evaluation and go to Step 7.

The Evolutionary Algorithm works through eight steps. The first step loads the data structures that are used during the evaluation process at step five and eight. These data structures are predefined lists which hold lexical items necessary to the evaluation. As an example, for an scholar environment, we could define the following lists:

List of Nouns: This list holds table names and their synonyms from a relational database filled by data about students, teachers and employees. *Student, Teacher, Secretary, Grade, Final Results* are an example of the content of this list. These nouns are classified as a *Ta* component in a basic sentence.

List of Selection Elements: This list holds lexical items that identify a request: *Select, Show, Make a Group, Who, Which* are examples of such selection elements. These elements are classified as the formal component *Ps* inside a basic sentence.

List of verbs: This list identifies verbs correlated to the domain. If we consider the scholar domain, some of the verbs we can identify are: *to study, to teach, to pass*, etc. If we take into account that the place where the student lives is important and it is part of the relational database, then the verbs *to live, to move* must be considered too. Support verbs, which are very used by natives in Brazilian Portuguese, are another kind of linguistic structure which identify an action, and must be put into this list. *Obteve sucesso* (got success), *foi reprovado* (did not pass) are some of these structures inside the scholar domain. Therefore, the linguistic structure *support verb + noun* is also included.

List of Fields: This list contains names and synonyms associated to a table field (column name) of the database, such as *student name*, *address*, *phone number*, etc.

List of Instances: This list contains data inside a table (*names*, *cities*, ...).

The last three lists described are important to identify the component R_l .

The number and the content of the lists depend on the domain and the database, each domain requires different lists which helps to identify (Ps, Ta, R) in a sentence.

The second step of the algorithm is responsible for tagging the grammatical classes for each word inside an input sentence. For instance, the sentence

(Students from Goiás or Students from Rio de Janeiro)

Alunos de Goiás ou alunos do Rio de Janeiro

is tagged as

Alunos N de PRP Goiás N ou CJ alunos N do PRP Rio de Janeiro N

In Brazilian Portuguese, the following tags are identified:

ADJ - Adjetivo (Adjective), *ADV* - Advérbio (Adverb), *ARTD* - Artigo definido (Defined Article), *ARTI* - Artigo indefinido (Undefined Article), *CJ* - Conjuação (Conjunction), *CPR* - Contração (Contractions), *ESTR* - Palavra estrangeira (Foreign word), *IN* - Interjeição (Interjection), *N* - Substantivo (Noun), *NUM* - Numeral (Number), *PART* - Particípio passado (Past Participle), *PRN* - Pronome (Pronoun), *PRP* - Preposição (Preposition), and *V* - Verbo (Verb).

These tags and the lists help the Evolutionary Algorithm to identify the formal structures Ps_m , Ta_m and R_l , when a complex sentence partition is performed for subsequent evaluation.

The third step generates the patriarch. This first individual of the population is created by cutting, using the symbol “/”, the sentence after each word; it means that the patriarch is the worst individual of the population, since it does not have any correct cut on its form. It is impossible to identify the formal structures Ps_m , Ta_m , R_l on the patriarch after tagging. In the following example, we show this first individual in which the cuts are characterized by the symbol “/”.

As an example, the patriarch with tags is

(Students from Goiás or Students from Pará)

Alunos N / de PRP / Goiás N / Ou CJ / Alunos N / do CPR / Pará N

C1	Students	/	from	/	Goiás	/	or	/	Students	/	from	Pará	1
C2	Students		from	/	Goiás	/	or	/	Students		from	Pará	5
C3	Students		from	/	Goiás	/	or		Students		from	Pará	8
C4	Students		from		Goiás	/	or		Students		from	Pará	10

Fig. 2. Possible chromosomes generated by patriarch mutations

Fig. 2 shows a set of possible mutations from the patriarch. This is performed by step 4 of the Evolutionary Algorithm. Notice the mutation process is performed by randomly inserting a symbol “/” in the individual’s chromosome or taking one off.

After a number of N mutations (N individuals generated), the algorithm goes to the evaluation step, where the lists of lexical items and tags are applied to evaluate each chromosome, identifying the formal structures from each cut (Ps_i , Ta_m and R_l) and assigning to these chromosomes an score value, as the next code below shows. It occurs from step 5 to 8, the algorithm searches for the best chromosome that presents a complex sentence broken into basic ones. This final choice is made by evaluating the total score of each chromosome in the last generation.

The evaluation of chromosomes are made by using the tags and lists of lexical items as heuristics to the Evolutionary Algorithm. These heuristics, as shown below, are in the form of rules that score each sequence of cuts. The last column in Fig. 2 above shows the score values to each chromosome after the evaluation done to each sequence between the symbols “/” in a sentence. This generation has **C4** as the best one.

```

IF   $\exists$ (Ls ) AND   $\exists$  (Ln) AND (  $\exists$  (Lf $\rightarrow$ Li) OR  $\exists$  (Lf $\rightarrow$ ADJ))
Format :=  Psi Ta1 R1!
Score :=  Score + 7
ELSE IF   $\exists$  (Ls) AND   $\exists$  (Ln) AND NOT(  $\exists$  (Lv))
    IF (Ln $\rightarrow$ Ls)
        Format :=  Ta1 Psi
        Score := Score + 5
    ELSE
        Format :=  Psi Ta1
        Score := Score + 5
    END IF
...
ELSE
    Format :=  $\Phi$ 
    Score := Score - 2
END IF

```

This code is an example of heuristic evaluation rules. Each **IF** statement is able to score each cut of a chromosome. The symbol $\exists$ is the logical quantifier “Exists” and the symbol $\rightarrow$ means that two linguistic objects are in sequence. Notice the logical operators (**AND**, **OR**, **NOT**) are also used. The linguistic objects used by the inference process shown above (Ls , Ln , Lv) are the lists of selection elements, nouns, verbs, ... Previously described. The tags (*ADJ*, *ADV*, *NUM*) of the sentences are also used by this process. Thus, for example, the first line of code means: if there is a term from the *List of Selection* ($\exists(Ls)$) and there is a term from the *List of Nouns* ($\exists(Ln)$) and, there is a sequence: *Fields* followed by *Instances* ($\exists(Lf \rightarrow Li)$) or there is a sequence (**Lf** $\rightarrow$ **ADJ**) where an element from *Lf* is followed by an *adjective*, then probably it has found a cut that fits in the sequence $Ps_i Ta_1 R_1!$ and this cut provides an additional score increment of $Score + 7$, where *Score* is the accumulated score of the chromosome. The increments of the heuristics were empirically chosen.

After a total score is calculated, for each chromosome, the best chromosomes, associated to the best and intermediate score values, are chosen. Many generations are

performed (steps 6 to 8) in order to improve the chromosomes, until the optimal (or the suboptimal) solution is achieved. Finally, the stack of nouns is filled.

4 The Translation Module

The second level of the system carries out the translation process. Its goal is to receive the basic sentences (extracted by the first module, previously described) and to associate each of them to SQL queries. Thus, it can solve the whole complex request emitted by a user. We could use many different methods to translate the basic sentences to an SQL query, but we also adopted a combinatorial approach to this stage. We could capture the sentences produced inside a domain via local grammars. They are represented by finite-state automata and can be built in modules, each of them presenting a semantic field. Fig. 3 depicts a finite-state machine able to capture questions about high grades at the scholar environment.

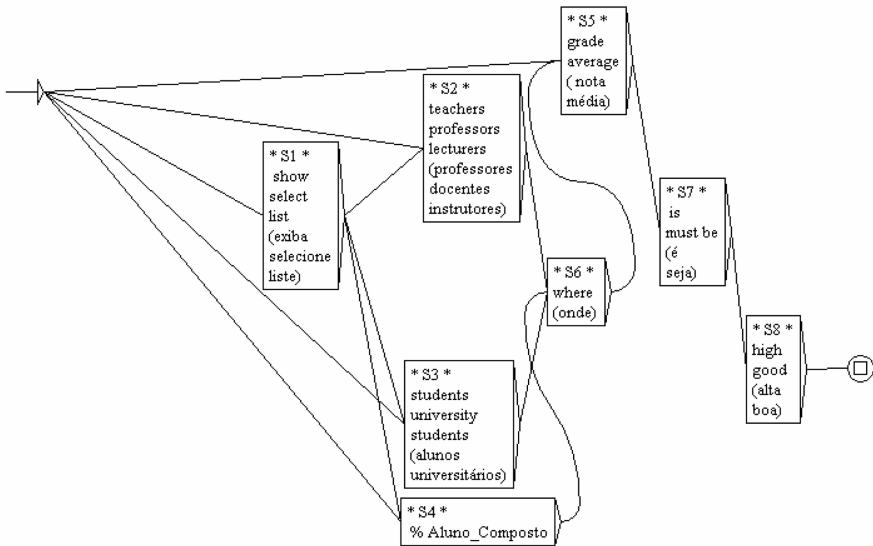


Fig. 3. Finite-state machine to capture Portuguese basic questions about high grades

Each grammar state has a list of lexical items (one or many words) that have similar meaning. The grammar in Fig. 3 identifies a sentence if, from an initial state, a graph search algorithm can reach a final state, having a basic sentence as input and an SQL query as output. This algorithm passes through the graph and it holds the label of the states $S_1 \dots S_n$ in a buffer, while passing among the states. Another buffer, if necessary, could store the content of some special states during the *parsing* process, it could hold proper names that would be used by the final SQL instruction. If the final state is reached, the sequence of states generates a unique key of states $S_1 \dots S_n$ that points to a unique SQL statement. Finally, the final set of records is built by the *intersection* among the sets of records generated by each simple sentence translation,

creating the final result. The grammar above is able to identify the following sentences:

- a) Show students where grade is high (*Exiba alunos onde nota é alta*)
- b) Grade must be good (*Nota seja boa*)

The state “%Aluno_Composto” is an “explosion state”, where more complex lexical items can be described. So, this state can be presented by another finite-state machine able to recognize other complex word combinations. For example,

- a) those geniuses (*aqueles gênios*)
- b) the most intelligent (*os mais inteligentes*)

The use of semantic rules, as previously described in section 3, can also be employed if the finite-state machine is not able to identify a basic sentence. So, if the machine does not reach a final state when a search algorithm is applied, a set of semantic rules could try to deduce the information inside a basic sentence. Then, a basic sentence can be recognized and translated by a finite-state machine applying a graph search algorithm or applying pattern matching on these machines. The match procedure compares the words of a sentence and the states of the grammar, the position of these words inside this sentence and the existence of grammar states through the basic sentence. The next code shows a semantic rule used to deduce a similar basic sentence from one that was not recognized by the graph search algorithm. This rule makes it possible to verify the existence of chained lexical items ($(S_1 \rightarrow (S_2 \text{ OR } S_3 \text{ OR } S_4) \rightarrow S_5 \rightarrow S_8)$), or the existence of grammar states (lines 2 and 4) through the sentence that presented a failure, where $S_1, \dots, S_n$ are states in Fig. 3. The rules could use the stack information to solve anaphoric references too (line 7).

1. If $((S_1 \rightarrow (S_2 \text{ OR } S_3 \text{ OR } S_4) \rightarrow S_5 \rightarrow S_8))$ Then
2. If $\exists (S_2)$ AND $(\text{NOT } \exists (S_3 \text{ OR } S_4))$ Then
3. Suggestion = Select teachers where grade is
 high (*Selecione professores onde nota é alta*)
4. Else If $\exists (S_3 \text{ OR } S_4)$ Then
5. Suggestion = Select students where grade is
 high (*Selecione alunos onde nota é alta*)
6. End If
7. Else If $\exists$ In STACK ('Students') AND $(\text{PRN} \rightarrow S_5 \rightarrow S_8)$ Then
8. Suggestion = Select students where grade is
 high (*Selecione alunos onde nota é alta*)
9. End If

5 Empirical Test and Results

A prototype was implemented to test the proposed methodology. By using a relational database composed by four tables and thirteen fields, a natural language interpreter was written. This prototype is able to answer questions about students that made a test to enter in a public university: *student grade values*, *address*, *student name*, *subjects* and the *name of the chosen university* are possible questions. An amount of 115

questions were applied. This set was composed by three classes: simple sentences, having just one constraint, and more complex sentences having two and three constraints (two or three basic sentences) combining different queries into one. We checked the correct translations by running the system for each sentence.

The Evolutionary Algorithm described in section 3 can be implemented in many different ways. One of the possible approaches, which produced a good result, employs an elitist strategy. Specifically, our implementation has the following features: the best solution found so far is preserved to the subsequent generation (elitism) and the other members of the population are K mutated chromosomes; a mutation is performed interchanging a symbol “/” by a blank space or vice-versa; only the chromosomes with a score higher than the half score value of the best individual are selected for mutation; less than 5 generations with 2000 individuals were sufficient to meet the requirements.

To the translation module, different algorithms for graph search can also be used. In our experiment, a variant version of the breath-first search algorithm was applied, where not all the descendent nodes from a patriarch node are expanded. Since the finite-state automata are deterministic machines and one lexical item does not appear in different nodes coming from the same parent node, the descendent nodes are opened by the search algorithm only until the lexical item searched is matched.

From an engineering point of view, the semantic rules were designed to be easily implemented directly using any programming language, and this fact contributes to an efficient and fast code.

In the proposed test bed, the prototype achieved a rate of 92% of correct translations.

Notice the time required by the evolutionary method is reduced. If we tried an exhaustive search to find all the possible combinations of “/” and blank spaces “ ” inside a complex sentence, the number of combinations would be very large. For example, if a request has 20 words, the number of possible combinations is $2^{19} = 524,288$ while our system generated at most 8000 combinations. Analyzing all possible combinations from the sentence to find the best set of cuts must take a very long time and it would be a problem to the user, since he or she wants the answer on-line. Although the number of combinations increases beside the number of words, the evolutionary approach needs less combinations, since it tries to improve more and more the solution (cuts) from the first individual after each population.

6 Conclusion

Our research tried to perform translation by means of a combinatorial approach.

We believe that, instead of using traditional methods based on several independent linguistic levels (lexicon, syntax rules, and semantics), trying to identify the possible combinations is an efficient alternative approach, since for natural languages it is impossible to identify general rules that cover the human linguistic productivity as proposed by generative grammar [5]. The less structured method permits an easier maintenance and control of the linguistic knowledge inside closed domains.

As a combinatorial approach is chosen, we are required to find algorithms and formalisms that are efficient and provide a good representation to this task. Notice the

search space to this problem is not so wide, since questions emitted by users from natural language interfaces are usually small sentences. Therefore, producing mutants (perturbed versions) until a good solution is found is not an expensive task. The time required by search algorithms to find a path through a graph is not long too, since the finite-state machines are small and do not have recursive paths.

System correctness is difficult to measure. Since the system relies on linguistic heuristics provided by semantic rules, correctness depends upon the rules written by a knowledge engineer. It is not possible to anticipate all the necessary rules at the first time the system is built. The linguistic rules produced inside a domain must be discovered through an investigation process, and it could be time-consuming, since the engineer must analyze the database, the language structures and failures emitted by the system. So we can say the process of identifying linguistic rules inside a domain is an evolutionary process too.

We suggest a combinatorial approach to the discourse level as a future work.

Combinatorial algorithms and traditional techniques, all together, provide a new alternative support to natural language processing.

References

1. Eiben, E.A., Smith, J.E.: *Introduction to Evolutionary Computing*. Springer, Heidelberg (2003)
2. Gross, M.: The Construction of Local Grammars. In: Roche, E., Shabes, Y. (eds.) *Finite State Language Processing*, pp. 329–352. MIT, Massachusetts (1997)
3. Roche, E., Shabes, Y.: Parsing with Finite-State Transducers. In: Roche, E., Shabes, Y. (eds.) *Finite State Language Processing*, pp. 241–278. MIT, Massachusetts (1997)
4. Gross, M.: On the relations between syntax and semantics. In: Keenan, E.L. (ed.) *Formal Semantics of Natural Language*, pp. 389–405. Cambridge University Press, Cambridge (1975)
5. Gross, M.: Les Bases Empiriques de la Notion de Prédicat Sémantique. *Langages* 63, 7–53 (1981)
6. Owda, M., Bandar, Z., Crockett, K.: Conversation-Based Natural Language Interface to Relational Databases. In: *IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology* (2007)
7. Weischedel, R.: Natural-Language Understanding at BBN. *IEEE Annals of the history of Computing* 28(1), 46–55 (2006)
8. Nerode, A.: *Logic for Applications*. Graduate Texts in Computer Science. Springer, Heidelberg (1997)
9. Silva, G.A.P.: Teoria dos Atos de Fala. *Cadernos do CNLF* 1, 259–263 (2005)
10. Pauw, G.D.: Evolutionary Computing as a Tool for Grammar Development. In: *GECCO*, pp. 549–560 (2003)

Attributes of Dynamic Combinatorial Optimisation

Philipp Rohlfshagen and Xin Yao

The Centre of Excellence for Research in Computational
Intelligence and Applications (CERCIA)
School of Computer Science, University of Birmingham
Birmingham B15 2TT, United Kingdom
P.Rohlfshagen@cs.bham.ac.uk, X.Yao@cs.bham.ac.uk

Abstract. The field of evolutionary computation has traditionally focused on static optimisation problems but recently, many new approaches have been proposed that adapt traditional evolutionary algorithms to deal with the task of tracking high-quality solutions as the search space changes over time. Algorithms developed specifically for dynamic domains have been tested on a wide range of different problems, including well-specified benchmark generators. However, the lack of theoretical results, a general omission of references to actual real-world scenarios, as well as a substantial emphasis on the continuous domain may divert attention away from some highly relevant issues. Here we review the state of the field and analyse dynamics in the combinatorial domain, using the subset sum problem as an example. It is shown that some of the assumptions underlying the development of new algorithms do not necessarily hold in the case of discrete optimisation. Furthermore, it is argued that more attention should be paid to the underlying dynamics and the impact of the representation used.

1 Introduction

The number of publications that address dynamic optimisation using evolutionary computation (EC) has increased significantly in recent years. In particular, several monographs ([1,2,3]) as well as edited volumes ([4]) have emerged that deal with this particular area of research. A typical motivation found in many publications may be summarised as follows: “Evolutionary algorithms (EAs) have traditionally been applied to static optimisation problems. However, many real-world problems are, in fact, dynamic.” Unfortunately, the majority of papers provide little (if any) additional context. Subsequently, authors often suggest new dynamic versions of well studied static optimisation problems such as the travelling salesman problem, job shop scheduling or multiple knapsack problems. Alternatively, publications make use of artificial benchmark problem generators, of which, however, only few exist. The three most widely used benchmarks are due to Branke (*moving peaks*; [1]), Morrison (*DF1*; [2]) and Yang (*XOR DOP*; [5]). Interestingly, only the latter is based on the combinatorial domain despite

the fact that almost all references to potential real-world scenarios are combinatorial in nature. The reasons for this tendency and its consequences are the focus of this paper.

The use of benchmark problems is vital in driving forward this relatively young field of research. However, the lack of theoretical results, the general omission of references to actual real-world scenarios, as well as the focus on the continuous domain may imply that a large amount of effort is directed at an academic problem that may only have limited relevance in the real world. Although this early work should be seen as a valuable starting point, the type of problems commonly approached seem to differ significantly from what one might expect in more realistic settings. This point has been raised previously [6] and here we continue along those lines. We provide a brief overview of the field of dynamic optimisation and address the issue of dynamic combinatorial optimisation in general. We propose a definition of such problems and evaluate some of the commonly held assumptions and attempt to demonstrate that these assumptions do not necessarily hold in the combinatorial domain. Complementary work has been carried out elsewhere [7,8] although here we aim to emphasise the notions of dynamics more precisely.

2 Dynamic Optimisation: A Brief Overview

In general terms, “a dynamic optimisation problem (DOP) is a problem that changes over time”, that is, the specifications of the problem are time-dependent. In numerous cases, this concept has been translated directly into new dynamic optimisation problems where an otherwise static problem is made dynamic by some alteration over time and solved using a newly proposed method. The general objective of dynamic optimisation is similar to the static case except that the parameters of the problem now depend on time t (adapted from Bosman, [9]):

$$\max_{\mathbf{x}(t)} \left\{ \sum_{t=0}^{end} f_{\gamma(t)}(\mathbf{x}(t)) \right\} \quad (1)$$

where $f_{\gamma(t)} : X \rightarrow \mathbb{R}$ is the fitness function with parameters $\gamma(t)$ and $\mathbf{x}(t)$ is the candidate solution at time t . A simple example of a dynamic problem is the moving sphere function, $f(\mathbf{x}) = g(|\mathbf{x}^* - \mathbf{x}|)$, where the dynamics, with magnitude θ and direction $\mathbf{v}$, are modeled as $\mathbf{x}^*(t+1) = \mathbf{x}^*(t) + \theta\mathbf{v}$ (see [10] for details).

Most approaches to DOPs attempt to reduce the computational complexity of the dynamic problem by “transferring knowledge from the past” [1]. Existing evolutionary algorithms (EAs) are modified to either maintain high levels of diversity throughout the optimisation process, create increased levels of diversity once a change has occurred, maintain a memory (implicit or explicit), employ specialised (mostly adaptive) genetic operators, or make use of multiple populations. In addition to these extensions, anticipation and prediction has been used recently in an attempt to exploit systematic patterns in the dynamics. Reviews of these techniques may be found in [11,12] or any of the monographs listed above.

2.1 Dynamic Benchmark Generators

The field of dynamic optimisation is still in its infancy, evident by the relative lack of theoretical work and the limited number of generally available benchmark problems. Benchmark problems are an essential tool in the development of new heuristics and the three most widely used benchmarks are *moving peaks*, *DF1* and *XOR DOP*. The former two are continuous in nature and model the search space as a “field of cones” [13], each of which may be controlled individually to model different ranges of dynamics. Although both benchmark generators are conceptually very similar, they are modeled in different ways and here we focus on *DF1*: in the two-dimensional case, the base function is given by $f(x, y) = \max_{i=1, N} [h_i - r_i \cdot \sqrt{(x - x_i)^2 + (y - y_i)^2}]$ where N is the number of cones, each at location (x_i, y_i) , and with height h_i and slope r_i . The initial morphology is randomly generated within the bounds specified by the user and the dynamics are modeled using a logistic function: $y(t) = \alpha y(t - 1)(1 - y(t - 1))$ where $\alpha \in (1, 4)$ is a constant.

There are numerous other continuous problems that have been used, including the simple moving sphere function mentioned earlier as well as a range of arbitrary functions with dynamic coefficients. The dynamics in these continuous problems are usually “fitness landscape oriented” which is made possible by the natural order of the continuous domain. The concept of fitness landscapes is also applicable in the combinatorial domain, although here, the neighbourhood depends on the encoding and the variation operators used. In the combinatorial domain, a fitness landscape $\mathcal{L} = (X, f, d)$ is defined by the set of solutions X , a fitness function f and the distance function d [14]. The distance measure d , which determines the proximity of points in the n -dimensional space, is often defined as the smallest possible distance although in practical terms, this depends on the variation (genetic) operators μ of the algorithm (in other words, it is impossible to define the notion of a fitness landscape in the combinatorial domain without regard to the algorithm used). The *XOR DOP*, initially proposed by Yang in [5], circumvents this algorithm-dependency to allow precise control over the frequency τ and the severity of change ρ by preserving the search space and rotating the candidate solutions by some degree prior to every function evaluation.

The *XOR DOP* allows to generate a dynamic version of any static binary problem: given a static fitness function $f(\mathbf{x})$, where $\mathbf{x} \in \{0, 1\}$, its dynamic equivalent is simply $f(\mathbf{x}(t) \oplus \mathbf{m}(k))$ where $\oplus$ is the bit-wise exclusive-or (XOR) operator and $t = 1, 2, \dots$ denotes the generation index. The period index $k = \lceil t/\tau \rceil$ is determined by the frequency of change, τ , again measured in generations. The vector $\mathbf{m}(k) \in \{0, 1\}$, initially $\mathbf{m}(1) = \mathbf{0}$, is a binary mask for period k , generated using $\mathbf{m}(k) = \mathbf{m}(k - 1) \oplus \mathbf{p}(k)$ where $\mathbf{p}(k) \in \{0, 1\}$ is a randomly created template for period k that contains exactly $\lfloor \rho n \rfloor$ ones. It follows that $\rho \in [0, 1]$ controls the magnitude of change. Usually both τ and ρ are kept constant throughout the execution of the algorithm.

These benchmarks allow full control over numerous aspects of the dynamics but have been criticised by Ursem et al. as “no research has been conducted to

thoroughly evaluate how well they reflect characteristic dynamics of real-world problems.” [6, p 1]. The authors further note that such benchmarks focus only on how the landscape changes and not the underlying dynamics themselves. In dynamic combinatorial optimisation, a strict landscape-oriented model is usually not applicable as the dynamics affect the parameter space and resulting changes in the fitness landscapes are dependent on f and μ . In the worst case, observed dynamics may be entirely uncorrelated to the underlying rules that generate them (see next section). This also implies that parameters such as ρ do not necessarily reflect the hardness of the dynamic problem and further theoretical analysis is required to gain a better understanding of what these benchmarks actually test. Finally, it is also worth noting that none of these benchmark problems consider constraints, which may have a significant impact on the performance of an algorithm.

3 Dynamic Combinatorial Optimisation

So far, a DOP has been defined simply as $f_{\gamma(t)}(\mathbf{x}(t))$. Here we extend this definition to place more emphasis on the actual dynamics and to distinguish between the dynamics in the parameter space and their impact on the fitness landscape. We follow the definition of a combinatorial optimisation problem by Garey and Johnson [15]: a combinatorial optimisation problem Π (either maximisation or minimisation) consists of the following three parts:

- a set S_{Π} of instances
- a finite set of candidate solutions $X_{\Pi}(I)$ for each instance $I \in S_{\Pi}$
- a function f_{Π} that assigns a positive rational number $f_{\Pi}(I, \mathbf{x})$ to each instance $I \in S_{\Pi}$ and each candidate solution $\mathbf{x} \in X_{\Pi}(I)$

Using this notation, a time-dependent (dynamic) optimisation problem (Π, T_{Π}) may be defined as $f_{\Pi}(I(t), \mathbf{x}(t))$ where $\mathbf{x}(t) \in X(t)$ ¹ is the candidate solution at time t for instance $I(t)$ and $I(t+1) = T(I(t), t)$ where $T : I \times \mathbb{N} \rightarrow I$ (also see [16]). We assume that the time t is discrete and we equate a single unit of time with a single function evaluation. The dynamics may be viewed as a trajectory through the space of all instances over time (state space) and this trajectory is described by T_{Π} , sometimes called the *meta-dynamics*. There are, of course, numerous other possible definitions but we believe that this definition highlights the role of the dynamics and the parameters and is thus a good starting point.

A dynamic problem may thus be seen as a discrete dynamical system and the algorithm’s task is to follow the orbit of successive global optima in the search space. It is important to stress that the dynamics affect the parameter space and corresponding changes in the fitness landscape depend on the algorithm used and the objective function. Rothlauf [17] provides a comprehensive description of encodings and defines two functions that map from genotypes to phenotypes and

¹ The space X is time-dependent if the dynamics affect the dimensionality and/or domain of the problem.

from phenotypes to fitness values: $f_g(\mathbf{x}_g) : X_g \rightarrow X_p$ and $f_p(\mathbf{x}_p) : X_p \rightarrow \mathbb{R}$ where $f = f_p \circ f_g = f_p(f_g(\mathbf{x}_g))$. As shown previously, a fitness landscape may be defined as $\mathcal{L} = (X, f, d)$ which, in practical terms, often translates to $\mathcal{L} = (X, f_p \circ f_g, \mu)$. Therefore, the dynamics of the problem affect the fitness landscape $\mathcal{L}$ which, in turn, is “observed” by the algorithm through $f_p \circ f_g$. If we define a distance measure $d_I : I \times I \rightarrow \mathbb{R}$ and a second distance measure $d_{\mathcal{L}} : X \times X \rightarrow \mathbb{R}$, we are essentially interested in whether $d_I \propto d_{\mathcal{L}}$. It is challenging to define a meaningful distance measure for the fitness landscapes. Here, we simply investigate the distance between the global optima, that is: $d_{\mathcal{L}}(\mathcal{L}(i), \mathcal{L}(j)) = d_H(\mathbf{x}^*(i), \mathbf{x}^*(j))$ where $d_H(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n |x_i - y_i|$ is the Hamming distance between two binary vectors. The distance measure for the instances is defined in section 3.1 following the description of the subset sum problem.

Intuition suggests that changes in the parameter space may have an arbitrary impact on the fitness landscape depending on representation and other factors. Consider the following simple 0/1 knapsack problem which may be defined as follows: given a vector of values $\mathbf{v} \in \mathbb{N}_+^n$, a vector of weights $\mathbf{w} \in \mathbb{N}_+^n$ and a positive integer $c > 0$, the objective is $\max\{\sum_{i=1}^n v_i x_i\}$ subject to $\sum_{i=1}^n w_i x_i \leq c$. The following instance illustrates how the distance between instances may be entirely un-correlated to the distance of the resulting global optimum:

Items	1	2	...	n-1	n
$\mathbf{w}(t)$	1	1	...	1	$c+1$
$\mathbf{v}(t)$	1	1	...	1	∞

where $c = n$. In this case, the optimal solution $\mathbf{x}^*(t) = 111 \dots 10$ with fitness $f(\mathbf{x}^*) = c$ changes to $\mathbf{x}^*(t+1) = 000 \dots 01$ with $f(\mathbf{x}^*(t+1)) = \infty$ given a change of 1 to the weight (i.e. $w_n(t+1) = c$) of the n th item. On the other hand, increasing the weight of the n th item by an arbitrary amount does not affect the global optimum at all. The manifestation of the dynamics thus depends on many factors, including the objective function as well as the representation and variation operators used.

This phenomenon has been noted frequently in the literature. Branke, for example, notes that in the classical dynamic $n = 17$ knapsack problem [18] with varying capacity $c(t)$, optimal solutions may become infeasible if $c(t+1) < c(t)$ and hence it seems unlikely that information from previous time steps may be reused [19]. Similarly, Yamasaki et al. [20] note that in binary optimisation problems, small changes in the parameters may result in (disproportionally) large changes in the objective function. This property generally contradicts the assumption that changes from one state to the next are smooth and is probably responsible for the focus on the continuous domain. However, if the majority of real-world scenarios are combinatorial in nature, and almost all examples in the literature refer to such cases, one cannot simply ignore this issue. Instead, one needs to analyse the impact of the encoding on the dynamics (which has already been done to some extent; see below) and take the dynamics of the problem into account explicitly.

3.1 Combinatorial Search Space Dynamics

Here we investigate how the fitness landscape $\mathcal{L}$ actually changes given well-defined distance measures for instances of the *subset sum problem*, a special case of the 0/1 knapsack problem where $\mathbf{w} = \mathbf{v}$. Formally, the subset sum problem is defined as follows: given a vector of weights $\mathbf{w} \in \mathbb{N}_+^n$ and a positive integer $c > 0$, the task is to find a subset of weights the sum of which is as close as possible, without exceeding, to c :

$$\max\left\{\sum_{i=1}^n w_i x_i\right\} \quad \text{subject to} \quad \sum_{i=1}^n w_i x_i \leq c \quad (2)$$

An instance is thus defined as $I = (\mathbf{w}, c)$ and the distance between two instances may be defined simply as the sum of absolute differences of all parameters: $d_I(I(i), I(j)) = |c(i) - c(j)| + \sum_{k=1}^n |w_k(i) - w_k(j)|$. We concentrate on instances with $0 < w_i \leq 100$ and $0 < c \leq 100n$ for $n = 10$ (the space of all instances S_Π is thus of size $100^{11} = 10^{22}$). The distance measure for the fitness landscapes is simply the Hamming distance of successive global optima, $d_{\mathcal{L}}(\mathcal{L}(i), \mathcal{L}(j)) = d_H(\mathbf{x}^*(i), \mathbf{x}^*(j))$. We are interested in the relationship between d_I and $d_{\mathcal{L}}$ and attempt to establish if and when such correlations exist.

There have already been some studies that examine the dynamics in combinatorial problems with special emphasis on the role of the representation [6,7,8]. The studies by Branke et al. compared three different encodings, *binary*, *permutation* and *weight*, on a dynamic version of the multiple knapsack problem. The authors found that although the permutation coding achieves the highest solution quality in the stationary case, the weight coding dealt more efficiently with the dynamics. The binary coding did not perform well, mostly because of the changes to the feasible area of the search space by an alteration of c . However, such indirect encoding also bears some noticeable problems in regard to T_Π . Consider, for example, a simple knapsack problem of size n with a capacity c that alternates between two non-changing values. In this case, once T_Π has been uncovered, the problem is essentially solved as it is now possible to predict the next state precisely. The binary coding allows such an approach as the functional mapping from genotype to phenotype (binary to subset) preserves the structure of the dynamics. The permutation coding, on the other hand, is highly redundant (e.g., 1234 and 2134 may be equivalent) and may thus mask any patterns, forcing the algorithm to continuously re-optimize the problem. The significance of the choice of encoding in stationary environments is well known (e.g., Prüfer numbers) and similar studies are required for the dynamic case where the role of the encoding seems even more significant.

One of the motivations behind the field of dynamic optimisation is computational tractability: problems under consideration are usually NP-hard and optimising such problems from scratch is very costly. If Π is NP-hard, there exists at least one instance I that can only be solved in exponential time. Usually, a problem has classes of instances with varying degrees of complexity and numerous measures have been proposed to measure the degree of difficulty of an

instance. One such measure is the fitness distance correlation (FDC; [21]) which indicates how well the fitness of a point relates to its distance to the nearest global optimum:

$$\text{FDC}(\mathbf{f}, \mathbf{d}) = \frac{1}{\sigma_{\mathbf{f}} \cdot \sigma_{\mathbf{d}} \cdot m} \sum_{i=1}^m (f_i - \bar{f})(d_i - \bar{d}) \quad (3)$$

where σ is the standard deviation, $\bar{\cdot}$ denotes the mean and m is the number of points sampled. There are some well known problems in using such measures. Here, however, we use this measure to identify properties that may have an impact on the relatedness of successive environments and not as a measure of instance difficulty. In instances with low FDC, peaks of similar height are scattered randomly across the fitness landscape. Subsequently, if a change occurs and the fitness values of some elements $\mathbf{x} \in X$ change, there is an increased likelihood that the new global optimum will not be closely related to the old one. If, on the other hand, there is a strong FDC, the new global optimum is likely to be situated in the vicinity of the old one. This will be investigated next by means of some experiments.

4 Experimental Setup

We use a simple genetic algorithm (GA) to evolve different instances of the subset sum problem with different FDCs. The GA used is steady-state with a population of size 100 and uniform crossover with probability $p_c = 0.8$. The individuals are integer vectors of length $l = n + 1 = 11$, encoding $n = 10$ weights and the capacity. The mutation operator, with probability $p_m = 1/l$, replaces an entry with a randomly chosen one. Selection and replacement are done by a binary tournament. We evolve 30 instances for each value of $\text{FDC} \in \{0, 0.1, \dots, 1.0\}$ (a total of 330 instances) ensuring each instance has only one global optimum. The FDC is determined by sampling all $m = 2^n$ points, using a minimum distance of 1 bit. As objective function for the subset sum problem, we use $f(\mathbf{x}) = s(c - \sum \mathbf{w}) + (1 - s) \sum \mathbf{w}$ where $s = 1$ if the solution is feasible, 0 otherwise [22].

We then generate direct neighbours of these instances and compare the global optima. We define the mass ω_I of instance I simply as $\sum_{i=1}^n w_i$ and subsequently define the distance ρ as a percentage of ω_I . Here we investigate $\rho \in \{0.1, 0.2, \dots, 0.5\}$. We generate adjacent instances in two different ways for any given distance d . In the first case, d_u , we simply modify each entry in $\mathbf{w}$ by the same amount $d/(n+1)$ while in the second case, d_r , a randomly chosen entry is modified by an arbitrary amount.

5 Results

A summary of results is shown in Table 1. It is evident that the higher the FDC, the lower $d_H(\mathbf{x}^*(t), \mathbf{x}^*(t+1))$ for all distances investigated. This holds true for

Table 1. Left-hand side: d_r , right-hand side: d_u . Hamming-distances of global optima for different values of d_I (top row), given instances with different FDCs (columns 1 and 8). The bottom row shows the Pearson correlation coefficient (P) for FDC and d_I . The second number in each column is the difference in fitness of the new global optimum. Columns 7 and 13 show the correlation for $d_H(\mathbf{x}^*(t), \mathbf{x}^*(t+1))$ and d_I .

	0.1	0.2	0.3	0.4	0.5	P		0.1	0.2	0.3	0.4	0.5	P
0	4.1 180	4.2 221	4.5 205	4.4 238	4.7 245	0.90	0	4.5 236	4.4 237	4.8 209	4.2 175	4.2 197	-0.60
0.1	4.2 261	4.0 285	3.9 257	4.0 212	4.3 224	0.20	0.1	4.2 213	3.7 229	4.4 169	4.4 169	4.4 267	0.50
0.2	3.9 213	4.0 224	3.9 209	3.5 186	4.4 258	0.21	0.2	4.2 187	4.0 189	4.2 216	4.2 146	4.1 187	0.05
0.3	3.8 131	4.0 238	4.1 258	3.8 188	4.4 295	0.57	0.3	3.2 213	4.0 204	3.9 216	3.9 184	4.4 229	0.82
0.4	4.2 92	3.8 162	4.1 184	3.9 175	4.1 173	0.04	0.4	3.6 98	4.3 98	4.2 124	3.6 132	3.9 148	-0.02
0.5	3.3 86	3.8 97	4.0 110	3.9 129	4.3 155	0.93	0.5	3.6 89	3.3 101	3.6 131	3.9 88	4.1 148	0.85
0.6	3.2 70	2.7 77	3.4 85	3.5 88	3.0 107	0.23	0.6	3.2 60	3.1 77	3.5 92	2.9 64	2.9 81	-0.49
0.7	1.3 31	1.8 54	1.7 64	1.8 81	2.3 89	0.92	0.7	1.4 41	1.6 41	1.9 50	1.9 49	2.1 76	0.96
0.8	0.4 5	0.4 21	0.7 58	1.3 89	0.8 38	0.73	0.8	0.4 30	0.8 17	1.0 27	1.4 52	1.1 5	0.87
0.9	0.1 0	0.3 0	0.3 0	0.7 53	0.9 117	0.97	0.9	0.1 0	0.3 0	0.9 0	0.8 1	1.0 0	0.91
1	0.1 96	0.3 95	0.5 123	1.2 223	1.4 143	0.96	1	0.1 46	0.4 67	1.0 26	0.9 0	1.3 2	0.96
P	-0.92	-0.92	-0.90	-0.89	-0.91		P	-0.94	-0.92	-0.94	-0.94	-0.91	

both distance measures although d_u shows a slightly higher degree of correlation than d_r . This result confirms the intuition that the global optimum is more likely to be situated a larger distance away from the old one if peaks of similar height are distributed randomly across the fitness landscape. The table also shows the differences in fitness between the old and the new global optimum and again, there is a strong correlation between the FDC and the magnitude of change. Interestingly, the differences diminish as the FDC increases with the exception of FDC=1. This is likely to be caused by the constraints, small changes to which may cause drastic changes in fitness due to the objective function used.

The data also shows that small changes may have a similar impact to large changes when the FDC is very low (column 7 and 13). This is because even slight changes may cause a new global optimum to appear elsewhere in the fitness landscape, even if the difference in fitness between the old and the new optimum is very small. It is of course questionable whether the additional computational effort required to locate the new global optimum may be justified by such small improvements in fitness.

6 Conclusion

In dynamic combinatorial optimisation, the dynamics T_Π describe a trajectory through the space of instances S_Π . The dynamics affect the parameters of the problem and are mapped onto the fitness landscape by a fitness function $f_p(\mathbf{x}_p)$. In addition, as the objective function $f : X \rightarrow \mathbb{R}$ is a black-box, the algorithm “observes” these changes by means of a genotype-phenotype mapping $f_p \circ f_g$. In other words, both the objective function and the algorithm’s encoding and genetic operators determine how the dynamics emerge. It has been shown that, in the case of the subset sum problem, even slight changes to the parameters of the problem may cause disproportionately large changes to the corresponding

fitness landscapes. In particular, the magnitude of change in regard to the global optimum $\mathbf{x}^*$ is directly proportional to the FDC of $I(t)$. Thus, our results do not support the speculation that “for most real-world problems, however, it is hoped that changes are rather smooth and, thus, a lot can be gained by transferring knowledge from the past” [11, p310].

The importance of the representation has been highlighted: some encodings may mask systematic changes that may otherwise be exploited to predict future states. It is important to distinguish between the performance of an algorithm on the stationary version of a problem and its dynamic counterpart. It has been shown that there are cases in the combinatorial domain where small changes in the parameters may produce highly uncorrelated fitness landscapes. Nevertheless, as long as the underlying dynamics are non-random, there is the potential to develop algorithms that perform better than a re-start approach. Similarly, Yamasaki et al. [20] conclude that EAs may be employed successfully in the dynamic domain if the underlying dynamics are structured. It is important to distinguish between a problem Π and the dynamics T_Π that are imposed on it. A dynamic problem has to be seen as a pair (Π, T_Π) and for every problem, a range of different dynamics may exist.

The focus on a dynamic version of the subset sum problem reveals numerous aspects that so far have not received sufficient attention and we believe much more effort should be directed at the combinatorial domain in general, the impact of constraints, the structure of the underlying dynamics and the impact of the representation on the manifestation of the dynamics in the resulting fitness landscapes. There are numerous additional issues that were beyond the scope of this paper and in the near future, we hope to expand on the topics discussed in this paper and to extend our work to cover these aspects as well.

Acknowledgements

We would like to thank Per Kristian Lehre, Trung Thanh Nguyen and Xing Lining for helpful comments. This work was supported by EPSRC grant no. EP/E058884/1.

References

1. Branke, J.: Evolutionary Optimization in Dynamic Environments. Kluwer, Dordrecht (2002)
2. Morrison, R.W.: Designing Evolutionary Algorithms for Dynamic Environments. Springer, Berlin (2004)
3. Weicker, K.: Evolutionary algorithms and dynamic optimization problems. Der Andere Verlag (2003)
4. Yang, S., Ong, Y.S., Jin, Y. (eds.): Evolutionary Computation in Dynamic and Uncertain Environments. Springer, Heidelberg (2007)
5. Yang, S.: Non-stationary problem optimization using the primal-dual genetic algorithms. In: Sarker, R., Reynolds, R., Abbass, H., Tan, K.C., McKay, R., Essam, D., Gedeon, T. (eds.) Proceedings of the 2003 IEEE Congress on Evolutionary Computation, vol. 3, pp. 2246–2253 (2003)

6. Ursem, R.K., Krink, T., Jensen, M.T., Michalewicz, Z.: Analysis and modeling of control tasks in dynamic systems. *IEEE Transactions on Evolutionary Computation* 6(4), 378–389 (2002)
7. Branke, J., Salihoglu, E., Uyar, S.: Towards an analysis of dynamic environments. In: Beyer, H.G., et al. (eds.) *Genetic and Evolutionary Computation Conference*, pp. 1433–1439. ACM, New York (2005)
8. Branke, J., Orbayi, M., Uyar, S.: The role of representations in dynamic knapsack problem. In: Rothlauf, F., Branke, J., Cagnoni, S., Costa, E., Cotta, C., Drechsler, R., Lutton, E., Machado, P., Moore, J.H., Romero, J., Smith, G.D., Squillero, G., Takagi, H. (eds.) *EvoWorkshops 2006*. LNCS, vol. 3907, pp. 764–775. Springer, Heidelberg (2006)
9. Bosman, P.A.N., Poutrè, H.L.: Learning and anticipation in online dynamic optimization with evolutionary algorithms: the stochastic case. In: *Proceedings of the 2007 Genetic and Evolutionary Computation Conference*, pp. 1165–1172 (2007)
10. Rossi, C., Barrientos, A., del Cerro, J.: Two adaptive mutation operators for optima tracking in dynamic optimization problems with evolution strategies. In: *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pp. 697–704 (2007)
11. Jin, Y., Branke, J.: Evolutionary optimization in uncertain environment - a survey. *IEEE Transactions on Evolutionary Computation* 9(3), 303–317 (2005)
12. Branke, J.: Evolutionary approaches to dynamic environments - updated survey. In: *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, pp. 27–30 (2001)
13. Morrison, R.W., DeJong, K.A.: A test problem generator for non-stationary environments. In: *Congress on Evolutionary Computation*, vol. 3, pp. 2047–2053. IEEE, Los Alamitos (1999)
14. Merz, P.: Advanced fitness landscape analysis and the performance of memetic algorithms. *Evolutionary Computation* 12(3), 303–325 (2004)
15. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York (1979)
16. Wolpert, D.H., MacReady, W.G.: No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1(1), 67–82 (1997)
17. Rothlauf, F.: *Representations for genetic and evolutionary algorithms*. Springer, Heidelberg (2002)
18. Goldberg, D.E., Smith, R.E.: Nonstationary function optimization using genetic algorithms with dominance and diploidy. In: Grefenstette, J.J. (ed.) *Second International Conference on Genetic Algorithms*, pp. 59–68. Lawrence Erlbaum Associates, Mahwah (1987)
19. Branke, J.: Memory enhanced evolutionary algorithms for changing optimization problems. In: *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, vol. 3, pp. 1875–1882. IEEE, Los Alamitos (1999)
20. Yamasaki, K., Sekiguchi, K.K.M.: Dynamic optimization by evolutionary algorithms applied to financial time series. In: *Proceedings of the 2002 Congress on Evolutionary Computation* (2002)
21. Jones, T., Forrest, S.: Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In: *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp. 184–192 (1995)
22. Wong, R.L.: A genetic algorithm for subset sum problem. *Neurocomputing* 57, 463–468 (2004)

A Weighted Local Sharing Technique for Multimodal Optimisation

Grant Dick and Peter A. Whigham

Department of Information Science,
University of Otago,
Dunedin, New Zealand
`gdick@infoscience.otago.ac.nz`

Abstract. Local sharing is a method designed for efficient multimodal optimisation that combines fitness sharing, spatially-structured populations and elitist replacement. In local sharing the bias toward sharing or spatial effect is controlled by the deme (neighbourhood) size. This introduces an undesirable trade-off; to maximise the sharing effect, deme sizes must be large, but the opposite must be true if one wishes to maximise the influence of spatial population structure. This paper introduces a modification to the local sharing method whereby parent selection and fitness sharing operate at two different spatial levels; parent selection is performed within small demes, while the effect of fitness sharing is weighted according to the distances between individuals in the population structure. The proposed method, as tested on several benchmark problems, demonstrates a level of efficiency and parameter robustness that surpasses the basic local sharing method.

1 Introduction

Many real-world problems present complex fitness landscapes in which there may be many desirable optima. An evolutionary algorithm (EA) can be encouraged to find multiple solutions to multimodal fitness landscape through the adoption of *nicing methods*. Over the many years of EA research, several niching methods have been developed, with most drawing upon the analogies between optima in a fitness landscape and the concept of species emergence in biological populations [1]. Exemplary niching methods include: crowding [2], fitness sharing [3], and species identification [4,5,6].

Recently, a new method called *local sharing* (LS) was introduced that incorporates fitness sharing into the selection process of spatially-structured populations [7,8]. The combination of operators within local sharing enabled it to be applied to multimodal problems previously unsearchable by just fitness sharing or spatially-structured populations alone. However, the local sharing method required careful calibration of the deme (neighbourhood) size, as the large deme size needed for accurate sharing contradicted the typically small deme sizes needed to maximise the effect of population structure.

This paper presents a new method of local sharing that applies a weighting (based on topological distances) to the basic sharing operator. As will be shown,

this has the effect of increasing reliability with respect to discovering optima, without the need to determine an appropriate deme size per problem. Additionally, it will be shown that *weighted local sharing* (WLS) offers a level of parameter robustness not necessarily offered by other elitist niching methods.

The remainder of this paper is structured as follows: the local sharing method, and limitations in its current design, are described in Section 2, while Section 3 presents a simple extension to local sharing; Section 4 outlines the analytical framework for the paper, while the performance of weighted local sharing over several benchmark functions is analysed in Section 5 and Section 6. Finally, a brief discussion of the findings of this paper and a suggested path for future work is presented in Section 7.

2 Local Sharing

The roots of the local sharing method lie in the overall area of spatially-structured evolutionary algorithms (SSEAs) [9]. This approach replaces the unstructured *panmictic* populations typically found in EAs with a population structure that restricts mating to topologically close individuals. While they enable a degree of multimodal search to take place, traditional SSEAs cannot be applied reliably to all multimodal problems. The LS method was designed to increase the applicability of SSEAs to multimodal function optimisation [7,8]. Its distinguishing feature is the application of the fitness sharing [3] within the demes of an SSEA. When tested on a number of benchmark problems, LS was shown to be applicable to more problems than fitness sharing or an SSEA alone.

The general process for local sharing is outlined in Algorithms 1 and 2. There are three main steps. First, the process of local sharing is applied to each deme in the population. Then, the next generation is produced as per a normal SSEA. Finally, another round of local sharing is applied to the offspring population and the shared fitnesses of individuals at each location are compared. Any offspring that are not *strictly better* than the current individual do not get passed into the next generation. This provides a degree of elitism in the algorithm and helps to ensure that good solutions are not unduly lost during evolution.

The sharing process applied to each deme is the same as for a typical “global” sharing method. First, the distance between all individuals within the deme is measured. Then, the niche count of an individual i is calculated as:

$$f_{sh}(i) = \frac{f(i)}{\sum_j sh(d_{ij})} \quad (1)$$

where $sh(d_{ij})$ measures the similarity between individuals i and j and is defined as:

$$sh(d_{ij}) = \begin{cases} 1 - \frac{d_{ij}}{\sigma_{sh}}, & \text{if } d < \sigma_{sh}; \\ 0, & \text{otherwise..} \end{cases} \quad (2)$$

In this equation d_{ij} is a measure of the distance (either genotypic or phenotypic) between individuals i and j . The parameter σ_{sh} is used to implicitly create

```

input : A spatially-structured population.
output: The same population with correctly calculated shared fitness of individuals.
1 population  $\leftarrow \{\}$ ;
2 foreach location in space do
3   i  $\leftarrow$  population[location];
4   deme  $\leftarrow$  constructDeme(location);
5   m  $\leftarrow$  0;
6   foreach j  $\in$  deme do
7     m  $\leftarrow$  m + sh(dij);
8   end
9   fitnesssh(i) = fitness(i)/m;
10 end
11 return population;

```

Algorithm 1. Applying fitness sharing within the demes of a spatially-structure population

```

input : A given problem
output: A spatially-structured population of evolved candidate solutions to the problem.
1 population  $\leftarrow \{\}$ ;
2 foreach location in space do
3   population[location]  $\leftarrow$  initialIndividual();
4 end
5 while not done do
6   localSharing(population);
7   generation  $\leftarrow \{\}$ ;
8   foreach location in space do
9     deme  $\leftarrow$  constructDeme(location);
10    parents  $\leftarrow$  select(deme);
11    generation[location]  $\leftarrow$  breedOffspring(parents);
12  end
13  localSharing(generation);
14  foreach location in space do
15    if fitnesssh(generation[location])  $\leq$  fitnesssh(population[location]) then
16      generation[location]  $\leftarrow$  population[location];
17    end
18  end
19  population  $\leftarrow$  generation;
20 end
21 return population;

```

Algorithm 2. The local sharing method

niches within the fitness landscape and must be set to an appropriate value for the problem at hand. In Section 6 we will examine the sensitivity of this parameter in the context of local sharing.

Previous work has demonstrated that the deme size used to search a problem has a significant effect of the LS algorithm's ability to find multiple optima [8]. The determination of ideal deme size is somewhat conflicting; ideally the deme size should be large to ensure that the sharing operator performs accurately. However, a large deme size effectively negates the spatial effect of the population topology, so with respect to the SSEA element of the algorithm the deme size should be kept small. Also, the balance between sharing and spatial effects must be tuned per problem, as demonstrated by Figure 1. Here, two benchmark problems, *Shekel's Foxholes* and the massively multimodal deceptive problem (*MMDP*) are searched using LS under a range of deme sizes. As can be seen,

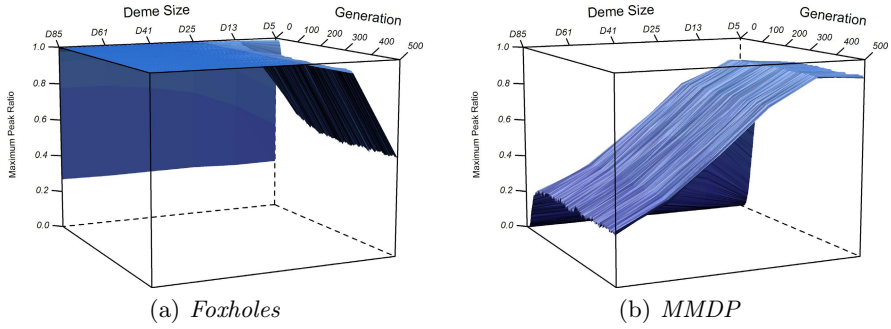


Fig. 1. The effect of varying the deme size within the local sharing method is problem-dependent

larger deme sizes lead to better maximum peak ratio performance (see Section 4) within the *Foxholes* problem. However, the opposite was true for the *MMDP* problem. Therefore, the determination of the appropriate deme size is heavily dependent on the problem being search. Previous work resorted to using “moderate” deme sizes as a compromise between the needs of the sharing and SSEA components of LS [8]. However, this compromise is undesirable, and a more reliable LS approach applicable over multiple problems is needed.

3 A Weighted Local Sharing Approach

The sharing component of local sharing requires a suitably large deme size. One way to achieve this would be to use an additional deme size for the sharing process, while maintaining the small, tight deme structure for parent selection. The two-deme approach has been used previously in SSEAs for function optimisation [10]. However, in this study, we will take a slightly different route. Parent selection will occur in a tight Von-Neumann neighbourhood¹ within a toroidal population structure. Then, the demic sharing process is replaced with one that measures the sharing effect over all population members and weights these measurements according to the topological distances between individuals. The algorithm for this method is outlined in Algorithm 3. This approach has the desired effect of introducing two scales of operation within space; the ratio between deme size and population size can be maximised, so the spatial effect introduced will be at its greatest. Meanwhile, the reach of sharing is extended throughout the population, so it can perform its job with a larger sample.

The weighted local sharing method described here requires a function to determine the weight to apply for a given pair of individuals at locations a and b .

¹ A Von-Neumann neighbourhood consists of the north, south, east and west neighbours of a given cell.

```

input : A spatially-structured population.
output: The same population with correctly calculated shared fitness of individuals.
1 population  $\leftarrow \{\}$ ;
2 foreach location a in space do
3   i  $\leftarrow$  population[a];
4   m  $\leftarrow$  0;
5   foreach location b in space do
6     j  $\leftarrow$  population[b];
7     m  $\leftarrow$  m + sh(dij)  $\times$  W(a, b);
8   end
9   fitnesssh(i) = fitness(i)/m;
10 end
11 return population;

```

Algorithm 3. Applying fitness sharing within the demes of a spatially-structure population

Here we will use a simple mapping (shown graphically in Figure 2):

$$W(a, b) = \left(\frac{1}{1 + d_{ab}} \right)^2. \quad (3)$$

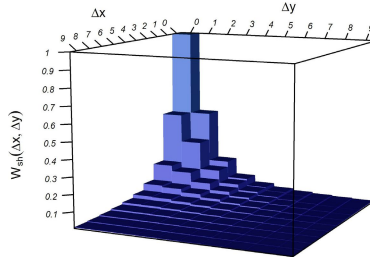


Fig. 2. A potential function for determining the weights applied to the sharing function in the local sharing method

4 Experimental Framework

The weighted local sharing method was compared against local sharing over a six benchmark problems. For a point of reference, standard fitness sharing and clearing were also applied to the test problems. The metric of interest was the maximum peak ratio (MPR) [4], which not only measures the niche discovery ability of a niching method, but also performs a quantitative measure of the method's ability to climb to the top of the peaks in the fitness landscape. The test functions used are listed in Table 1 and the reader is pointed to the relevant references for specific information.

Each algorithm was applied to each problem 100 times and the average MPR value was measured at the end of each generation. Population sizes and the niche radius for each problem are described in Table 2. For all problems, selection within the demes in LS was via roulette wheel selection, while clearing

Table 1. The test functions used in this paper

Key	Function Name	Range	Comments
<i>Foxholes</i>	Shekel's Foxholes [11]	$x, y \in [-65.536 : 65.535]$	25 optima of varying fitness
<i>MMDP</i>	Massively multimodal, deceptive problem [12]	30-bit bitstring	32 global optima, 5×10^6 deceptive optima
<i>MINHD</i>	Minimum distance function [13]	24-bit bitstring	27 optima of varying fitness, 2170 deceptive optima
<i>Shubert</i>	Shubert function [6]	$x \in [-10 : 10]^D$	$D \cdot 3^D$ global optima, many local optima

Table 2. The problem-specific parameters used in this paper. Values for these parameters were sourced from previous work[5,7,15].

	Foxholes MMDP MINHD			Shubert		
				2D	3D	4D
Pop. Size	289	676	1024	484	676	2025
Niche radius	8	6	3	0.4		

and global sharing used stochastic universal selection [14]. All problems used a bitstring representation with a simple binary-coded decimal mapping for the real-parameter problems. One-point crossover was applied with 100% probability. Bit-flipping mutation was applied at the rate of 0.002 per locus. For the standard LS approach, a simple Von-Neumann neighbourhood was used for all problems, except for the *Foxholes* problem which used a larger deme size of 25 individuals.

5 Results

The MPR performance on all algorithms on the six test problems is shown in Figure 3. Both the local sharing methods outperform the global sharing approach; additionally, even though it was using the same deme structures throughout all test problems, the WLS approach consistently outperformed standard LS. The performance of WLS was comparable to clearing on all the problems except for the higher-dimension Shubert problems; at this point, clearing is much faster at finding optima, but WLS eventually reaches a similar level of MPR performance. Because of its lack of elitist replacement strategies, sharing fails on all problems except for the relatively simple Foxholes problem.

6 Parameter Robustness

The WLS and clearing methods appear to offer good performance. The MPR performance suggests that they are able to locate more optima in the fitness landscape than other methods using similar population sizes [15]. However, the methods used in [15] are essentially free from parameterisation, whereas the experiments conducted here are under ideal parameter settings, so the comparison is not strictly fair. There is a need for algorithms to be robust to parameterisation if they are to be reliably applied to real-world problems. Clearing is already known to be sensitive to parameterisation [6,15]. The robustness of the WLS

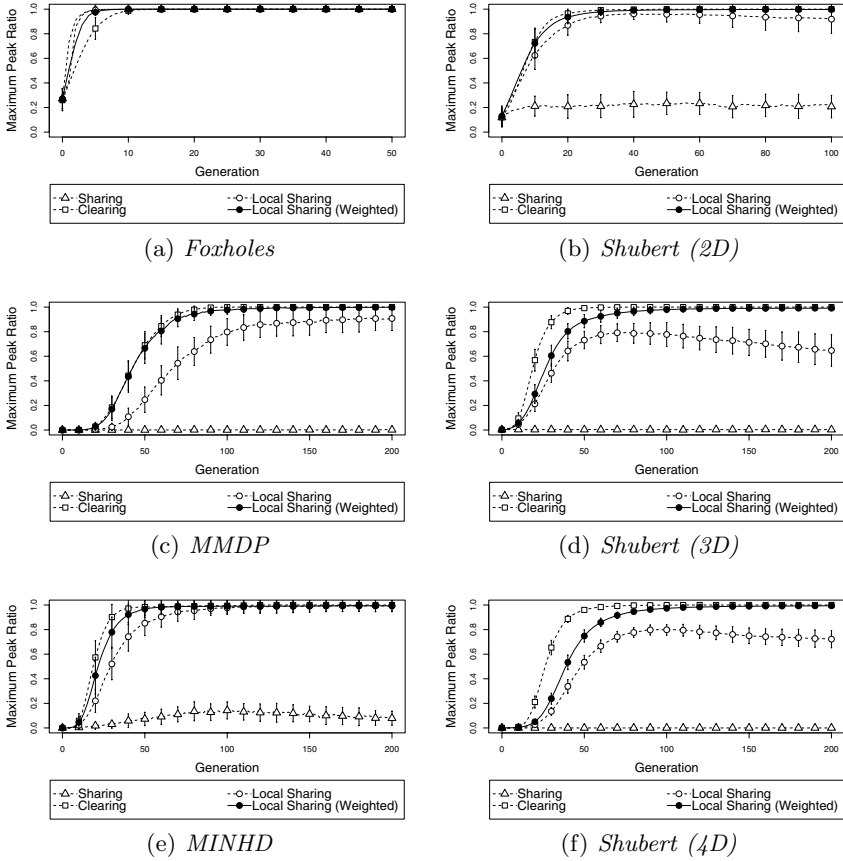


Fig. 3. The maximum peak ratio performance of weighted local sharing as compared to clearing, fitness sharing and standard local sharing. Error bars are represent one standard deviation around the measured value.

approach, however, is an unknown quantity as it has not been tested previously. Therefore, to test the parameter robustness of WLS, it was reapplied to all six test problems under a range of values for the niching radius. For a point of reference, similar tests were performed with clearing.

The MPR performance of the WLS method for these tests is shown in Figure 4 and the equivalent tests for clearing are presented in Figure 5. As can be seen, while the performance of clearing quickly drops off in direct response to an increase in niche radius, the performance of weighted local sharing is largely unaffected by the value of the niche radius. In particular, the *MINHD* and *2D-Shubert* functions are virtually unaffected. On the *Foxholes* problem, WLS performance starts to drop off when the niche radius reaches 20; at this point, the niche radius is large enough to overlap as many as 9 optima in the fitness landscape at any one time. Performance of the WLS approach on *MMDP* is slightly affected by

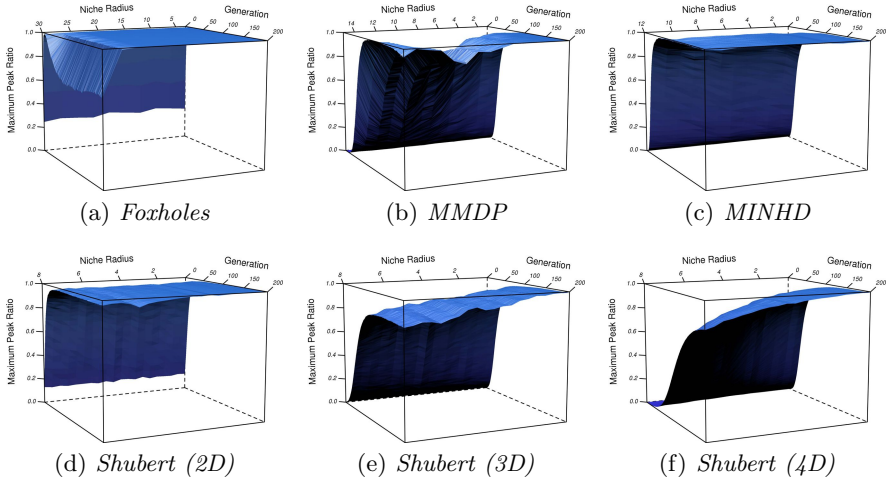


Fig. 4. The effect of varying niche radius of the maximum peak ratio performance of weighted local sharing

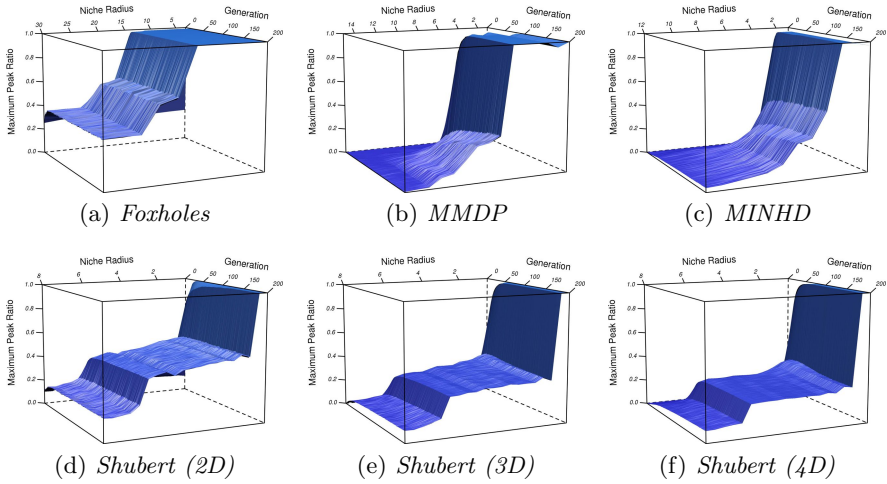


Fig. 5. The effect of varying niche radius on the maximum peak ratio performance of clearing

the value of the niche radius, however it still managed to return (on average) at least 80% of the optima in the fitness landscape for every tested value of the niche radius. The level of parameter robustness presented by WLS, coupled by the good MPR performance demonstrated in Section 5 suggest that one could apply this method to real-world problems with a reasonable level of confidence.

An exhaustive analysis of the reasons behind the relative parameter robustness of WLS are beyond the scope of this paper. However, we can briefly offer two theories that might help to explain the observed robustness. The first is that

both empirical and theoretical evidence suggests that the sharing method is in fact rather robust to moderate levels of niche overlap [16,17,18]. Therefore, it would be reasonable to assume that this robustness would carry over to the WLS method. Another theory behind the cause of parameter robustness could lie in the inherent parallel nature of local sharing. Typically, a global niching method will apply its operators once per generation over the entire population. However, local sharing applies a separate instance of sharing at each location in space; because each instance of sharing operates in isolation from the rest of the population, it is free to focus upon only a subset of the optima in the fitness landscape. If each deme concentrates on a unique combination of optima, a level of redundancy in the search would result and a larger number of niches would emerge when considered at a global level.

7 Conclusion

Local sharing is an novel approach to niching that applies niching at both the individual and population levels. The standard local sharing method, however, requires careful parameterisation to the given problem in order to operator at its best. This paper has presented a modification to the local sharing approach that uses topological distances between individuals to weight the sharing process. This approach, as tested here, can be applied to multiple problems using the same demic parameter values throughout. Additionally, the WLS approach offers a similar level of peak discovery to that of clearing, while at the same time offering a much greater level of parameter robustness with respect to the niching radius. However, the results presented here suggest that clearing is still a better climber of peaks than WLS; future work should investigate the use of methods to improve the climbing performance of WLS. Sharing is a “global” optimiser in that it is not efficient at climbing peaks [19]. Rather, it should be coupled with a strong local search method to complete the process. Spatially-structured EAs are easily augmented with local search techniques [20], so this could result in a powerful niching algorithm.

References

1. Mayr, E.: Populations, species and evolution; an abridgment of *Animal species and evolution*. Harvard University Press (1970)
2. Mahfoud, S.W.: Crowding and preselection revisited. In: Männer, R., Manderick, B. (eds.) *Parallel problem solving from nature 2*, pp. 27–36. North-Holland, Amsterdam (1992)
3. Goldberg, D.E., Richardson, J.: Genetic algorithms with sharing for multi-modal function optimisation. In: *Proc of the 2nd Int. Conf. on Genetic Algorithms and Their Applications*, pp. 41–49 (1987)
4. Miller, B.L., Shaw, M.J.: Genetic algorithms with dynamic niche sharing for multimodal function optimization. In: *International Conference on Evolutionary Computation*, pp. 786–791 (1996)
5. Pérowski, A.: A clearing procedure as a niching method for genetic algorithms. In: *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pp. 798–803 (1996)

6. Li, J.P., Balazs, M.E., Parks, G.T., Clarkson, P.J.: A species conserving genetic algorithm for multimodal function optimization. *Evolutionary Computation* 10(3), 207–234 (2002)
7. Dick, G., Whigham, P.A.: Spatially-structured evolutionary algorithms and sharing: Do they mix? In: Wang, T.D., Li, X., Chen, S.H., Wang, X., Abbass, H.A., Iba, H., Chen, G., Yao, X. (eds.) *SEAL 2006*. LNCS, vol. 4247, pp. 457–464. Springer, Heidelberg (2006)
8. Dick, G., Whigham, P.: Spatially-structured sharing technique for multimodal problems. *Journal of Computer Science and Technology* 23(1), 64–76 (2008)
9. Tomassini, M.: *Spatially structured evolutionary algorithms*. Springer, Heidelberg (2005)
10. Ishibuchi, H., Doi, T., Nojima, Y.: Effects of using two neighborhood structures in cellular genetic algorithms for function optimization. In: Runarsson, T.P., Beyer, H.G., Burke, E.K., Guervos, J.J.M., Whitley, L.D., Yao, X. (eds.) *PPSN 2006*. LNCS, vol. 4193, pp. 949–958. Springer, Heidelberg (2006)
11. Mahfoud, S.W.: *Niching methods for genetic algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL, USA, IlliGAL Report 95001 (1995)
12. Goldberg, D.E., Deb, K., Horn, J.: Massive multimodality, deception, and genetic algorithms. In: Männer, R., Manderick, B. (eds.) *Parallel Problem Solving from Nature*, vol. 2, pp. 37–46. Elsevier Science Publishers, B. V, Amsterdam (1992)
13. Horn, J., Goldberg, D.E.: Genetic algorithm difficulty and the modality of fitness landscapes. In: Whitley, L.D., Vose, M.D. (eds.) *Foundations of Genetic Algorithms* 3, pp. 243–269. Morgan Kaufmann, San Francisco (1995)
14. Baker, J.E.: Reducing bias and inefficiency in the selection algorithm. In: Grefenstette, J.J. (ed.) *Genetic Algorithms and their Applications (ICGA 1987)*, pp. 14–21. Lawrence Erlbaum Associates, Hillsdale (1987)
15. Bird, S., Li, X.: Adaptively choosing niching parameters in a PSO. In: Keijzer, M., Cattolico, M., Arnold, D., Babovic, V., Blum, C., Bosman, P., Butz, M.V., Coello Coello, C., Dasgupta, D., Ficici, S.G., Foster, J., Hernandez-Aguirre, A., Hornby, G., Lipson, H., McMinn, P., Moore, J., Raidl, G., Rothlauf, F., Ryan, C., Thierens, D. (eds.) *2006 Genetic and Evolutionary Computation Conference (GECCO 2006)*, vol. 1, pp. 3–10. ACM Press, Seattle (2006)
16. Deb, K., Goldberg, D.E.: An investigation of niche and species formation in genetic function optimization. In: Schaffer, J.D. (ed.) *Proc. of the Third Int. Conf. on Genetic Algorithms*, pp. 42–50. Morgan Kaufmann, San Mateo (1989)
17. Horn, J.: *The Nature of Niching: Genetic Algorithms and the Evolution of Optimal, Co-operative populations*. PhD thesis, University of Illinois at Urbana Champaign (1997)
18. Watson, J.P.: A performance assessment of modern niching methods for parameter optimization problems. In: Banzhaf, W., Daida, J.M., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M.J., Smith, R.E. (eds.) *Proceedings of the 1999 Conference on Genetic and Evolutionary Computation (GECCO 1999)*, pp. 702–709. Morgan Kaufmann, San Francisco (1999)
19. Mahfoud, S.W.: A comparison of parallel and sequential niching methods. In: Eschelman, L. (ed.) *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp. 136–143. Morgan Kaufmann, San Francisco (1995)
20. Mühlenbein, H.: Parallel genetic algorithms, population genetics and combinatorial optimization. In: Schaffer (ed.) *Proceedings of the Third International Conference on Genetic Algorithms (ICGA 1989)*, pp. 416–421. Morgan Kaufmann, San Francisco (1989)

Hybrid Genetic Programming for Optimal Approximation of High Order and Sparse Linear Systems^{*}

Jing Liu, Wenlong Fu, and Weicai Zhong

Institute of Intelligent Information Processing, Xidian University, Xi'an, China
neouma@163.com

Abstract. A Hybrid Genetic Programming (HGP) algorithm is proposed for optimal approximation of high order and sparse linear systems. With the intrinsic property of linear systems in mind, an individual in HGP is designed as an organization that consists of two cells. The nodes of the cells include a function and a terminal. All GP operators are designed based on organizations. In the experiments, three kinds of linear system approximation problems, namely stable, unstable, and high order and sparse linear systems, are used to test the performance of HGP. The experimental results show that HGP obtained a good performance in solving high order and sparse linear systems.

1 Introduction

Since J. R. Koza extended John Holland's Genetic Algorithm (GA)^[1] to Genetic Programming (GP)^[2], GP has been widely applied to different applications. There are two main research directions in GP, namely standard GP^[2] and Linear structure GP (LGP)^[3]. LGP is based on the principle of register machines. In recent years, many variants of LGP have been proposed, such as Multi-expression Programming (MEP)^[4], Gene Expression Programming (GEP)^[5], Cartesian Genetic Programming (CGP)^[6]. An MEP gene encodes a terminal or a function symbol, and the gene that encodes a function includes pointers towards the function arguments. A CGP node contains a function symbol and pointers towards the nodes representing function parameters. Each CGP node has an output that may be used as input for another node. A node in MEP is a terminal symbol or an expression. GEP genes are composed of a head and a tail. The head contains both functions and terminals symbols. The tail contains only terminal symbols. Based on these variants of LGP, we proposed a new variant of LGP, namely Hybrid Genetic Programming (HGP), and applied it to optimal approximation of high order and sparse linear systems.

^{*} This work was supported by the National Natural Science Foundations of China under Grant 60502043, 60872135, and 60602064, the Program for New Century Excellent Talents in University of China under Grant NCET-06-0857, the National High Technology Research and Development Program ("863" program) of China under Grant 2006AA01Z107, and the Natural Science Research Project of Shaanxi, China.

Approximation of linear system models is an important task in the simulation and controller design for complex dynamic systems^[7]. In the attempt at solving model approximation problems, numerous methods, such as Multiagent Genetic Algorithm (MAEA)^[8], Differential Evolution Algorithm (DEA)^[9], and other methods^[10], have been proposed. They can be grouped into two major categories, namely the performance-oriented and the nonperformance-oriented approaches. In these methods, one hypothesis is a known model and the other one is the range of these parameters related to the known model. Due to these two hypotheses, these methods are hard to find a better solution for optimal approximation of high order and sparse linear systems.

HGP proposed in this paper can find an approximate model for linear systems automatically and has no relations with the parameters. Thus, it is different from the traditional methods that use a known model. Because HGP can find an approximate model automatically, it can be easily applied to optimal approximation of high order and sparse linear systems.

The rest of this paper is organized as follows: Section 2 describes the model approximation problem of linear systems. Section 3 introduces HGP algorithms for optimal approximation of linear systems. Section 4 gives the experimental results on three kinds of linear system approximation problems and performance comparisons. Finally, Section 5 concludes this paper.

2 Model Approximation Problem of Linear Systems

In the control community, it is well known that a complex dynamic system can be adequately approximated by a low-order transfer function with a time delay. Given a high-order rational or irrational transfer function $G(s)$, it is desired to find an approximate model of the form in (1) such that $H_m(s)$ contains the desired characteristics of $G(s)$.

$$H_m(s) = \frac{b_0 + b_1s + \cdots + b_{m-1}s^{m-1}}{a_0 + a_1s + \cdots + a_{m-1}s^{m-1} + s^m} \cdot e^{-\tau_d s} \quad (1)$$

MAEA^[8] and DEA^[9] aim at finding an optimal approximate model $H_m(s)$ such that the frequency-domain L^2 -error performance index in (2) is minimized, where the frequency points, ω_i , $i=0, 1, 2, \dots, N$, and the integer N taken a priori.

$$J = \sum_{i=0}^N |G(j\omega_i) - H_m(j\omega_i)|^2 \quad (2)$$

The problem of minimizing J given in (2) is an optimal parameter selection problem. The optimal parameters $a'_i s$, $b'_i s$, and τ'_d of the approximate model in (1) can be found. Traditional methods must depend on the model and set the optimal parameters in a range. However, the model cannot be set with ease and the parameters need to be located in a range when $G(s)$ is a high order and sparse linear system. HGP can automatically create an optimal model that possibly not contains all the parameters. The parameters of the model are independent of HGP.

3 Hybrid Genetic Programming

3.1 Cell and Organization in HGP

HGP uses prefix expression. In prefix expression, “ $(s+3) \times 5$ ” is represented as “ $s3+5 \times$ ”. When evaluating this expression, “ s ” and “ 3 ” is not active until “ $+$ ” is fetched. We cannot destroy the special structure if we want to perform operators, such as crossover, mutation, on the expression. For example, the expression “ $s3+\times 5$ ” is illegal. On the basis of MEP, GEP and CGP, the rules in HGP are designed as follows:

- (a) Head node only contains a terminal, and other nodes contain a terminal and a function;
- (b) When a terminal is a variable, the variable includes its exponent. When a terminal is a real number, it is perturbed;
- (c) A cell consists of Head node and other nodes;
- (d) Evaluate a cell from Head node to other nodes in order;
- (e) Cell is a special terminal, and an organization consists of the nodes containing the special terminal.

In rule (a), it is similar to MEP and CGP that a node contains a terminal and a function. While node in MEP and CGP contains index, node in HGP dose not contain. The rule (c) and (d) is similar to GEP, but in GEP, node only contains a terminal or a function, and a GEP chromosome contains a head and a tail. The rule (b) and rule (e) are new rules. HGP simply designs node and then is the simplest in evaluating among these methods.

A cell and an organization are shown in Fig. 1. The expression “ $(s+3) \times 5$ ” is encoded to a cell, and the expression

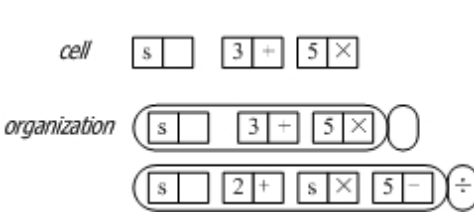


Fig. 1. A cell and an organization in HGP

“ $\frac{(s+3) \times 5}{((s+2) \times s) - 5}$ ” is encoded to an organization.

Simple problems can be solved with Cell. But it is hard to exactly describe a solution for some complex problems only with cell, such as

$$\frac{1000 + 10s^5 + \sin(s)}{s^2 \times \cos(s) + 20s^3 + 100s + 1}.$$

Because a cell is impractical to describe some complex functions, we borrow the idea from [11] and introduce the concept of “organization” into rule (e). Because organizations are composed of cells, they can undergo division and combination.

In this paper, we let an organization consist of only two cells because an optimal model usually consists of numerator and denominator. Head “node” (cell) is called numerator and the other “node” whose function only can be “ $\div$ ” is called denominator. If we take a delay time into account, a parameter, namely *delay* is added into the organization. The fitness of an organization is J .

The function set in a cell is $\{+, -, \times\}$ and the terminal set is $\{variable, real\}$, where *variable* consists of s and its exponent which is a positive integer, e.g. *variable*(s , 10)

denotes s^{10} . *real* is a real number and initialized by a randomly positive number in the range of [0, 10.0]. According to the relation of some operation symbols in [12], such as “+” and “-” have the same function and so do “ $\times$ ” and “ $\div$ ”, we do not take “ $\div$ ” into the function set. Because real numbers are positive when are initialized, operation symbol “-” is reserved.

3.2 Operators in HGP

Search operators used within HGP algorithm are crossover, mutation and study.

A) Crossover Operator

This operator is randomly performed on two organizations. There are four kinds of crossovers: In Crossover 1, the numerators in two organizations are crossed. In Crossover 2, the denominators in two organizations are crossed. In Crossover 3, both the numerators and the denominators in two organizations are crossed. In Crossover 4, the numerators or the denominators in two organizations are exchanged for each other. In each cell, a start node is randomly chosen, and nodes from the start node to end are exchanged between two nodes. An example for Crossover 1 is shown in Fig. 2. Two offspring, namely *Offspring₁* and *Offspring₂* are obtained by crossover between *Organization₁* and *Organization₂*. If *Offspring₁* is better than *Organization₁*, *Offspring₁* will survive and *Organization₁* will die, or vice versa. *Offspring₂* competes with *Organization₂*.

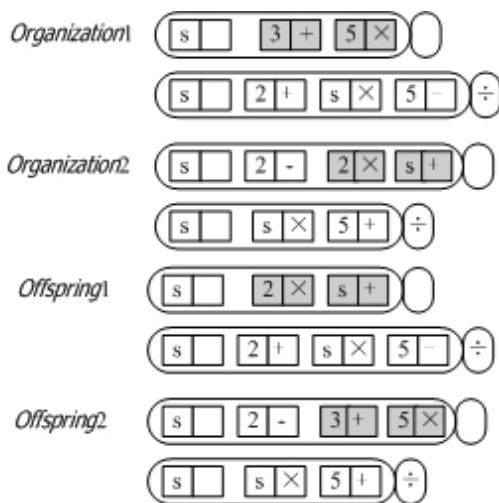


Fig. 2. An example for Crossover 1

B) Mutation Operator

This operator can add a node and delete a node, change the function in a node, and delete a sub-cell and add a sub-cell. Three kinds of mutations are used. In Mutation 1, two new organizations are produced by adding a node and deleting a node. In Mutation 2, a new organization is produced by changing the function in a node. In Mutation 3, two new organizations are produced by deleting a sub-cell and by adding a new sub-cell. An example for Mutation 1 is shown in Fig. 3. *Offspring₁* is obtained by deleting a node in *Organization* and *Offspring₂* is obtained by adding a new node to *Organization*. The best organization from the offspring and *Organization* would survive, and the others die.

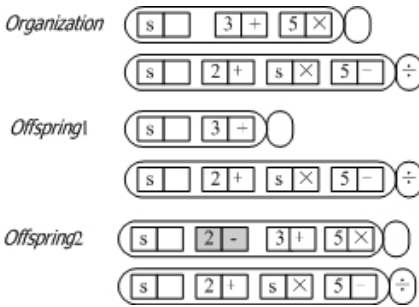


Fig. 3. An example for Mutation 1

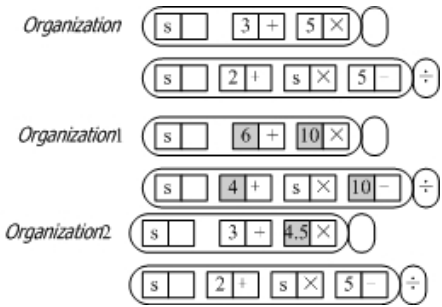


Fig. 4. An example for Study operator

C) Study Operator

This operator is designed based on ontogeny, and can change the value of the real numbers in an organization. For instance, in Fig. 4, *Organization*₁ is obtained by all real numbers of *Organization* increased 1 time and *Organization*₂ is obtained by “5” in a node of *Organization* decreased 0.5.

For example, when cell is as a solution for finding the expression “ $h=3s$ ”, a cell is $(s)(1.2, \times)(4.5, +)$. The number “1.2” increases to “3” and the number “4.5” decreases to “0”, and then the cell is the right solution. Generally speaking, the proportion of nodes including numbers in a cell is higher than 30%, which protects the building block from destroying. An embryo organization may be not better than its parent. We can get a better model after crossover or mutation operation, but the fitness is not good because of incorrectness real numbers. Study operation can cure this problem and accelerate the searching speed. Thus, we let an organization studying t_s times. The algorithm for Study operator is shown in Algorithm 1.

Algorithm 1. Study Operator
Function: <i>Study</i> (<i>Organization</i>)
Input: An organization
Output: An organization
Step 1. <i>study_time</i> =0;
Step 2. If the fitness of the organization is less than a constant, go to Step 4;
Step 3. Change all real numbers in the organization, and go to Step 5;
Step 4. Randomly select a real number in the organization, and change it;
Step 5. If a delay time is taken into account, the value of the delay time will be randomly increased or decreased by a number less than 0.1;
Step 6. If the fitness is not decreased, the real number changed returns the original value;
Step 7. <i>study_time</i> = <i>study_time</i> +1, if <i>study_time</i> < t_s , go to Step 2, otherwise stop.

3.3 Algorithm for Optimal Approximation of High Order and Sparse Linear Systems

Offspring competes with parents after studying, and better one survives with the idea of evolution strategy. Because Study operator is used, more embryos are needed. HGP algorithm is given in Algorithm 2 in details. Where *MAX_GENERATION* is the maximum number of generations, *M* is the size of population, and *pop[i]* is the *i*th organization in the population. When calculating the fitness, there are four points need to be noted: (1) If a cell has junk nodes, such as “1-1”, one of these nodes is deleted; (2) If the value of the denominator is 0, the denominator is mutated until the value of the denominator is not 0; (3) If the fitness is larger than a threshold value (in this paper, the value is equal to 1e+100), the numerator and the denominator are mutated until the fitness is less than the threshold value; (4) If the order of a cell is larger than the maximum order, the nodes including variable and “×” will not change the functions until the order is not larger than the maximum order.

Algorithm 2. Hybrid Genetic Programming

- Step 1. Initialize the population and calculate the fitness of each organization;
 - Step 2. Let *Best* be the organization whose fitness is minimum in the population;
 - Step 3. Let *Generation* be 0;
 - Step 4. Randomly arrange the organizations in the population;
 - Step 5. Let *i* be 1;
 - Step 6. Perform Crossover operator on *pop[i]* and *pop[i+1]*;
 - Step 7. Perform Mutation operator on *pop[i]* and *pop[i+1]*;
 - Step 8. *i=i+2*, and if (*i*<*M*), go to Step 6;
 - Step 9. If the minimum fitness in the population is less than that of *Best*, *Best* is set to the organization with minimum fitness in the population;
 - Step 10. Perform Study operator on *Best*;
 - Step 11. *Generation=Generation+1*, and if (*Generation*<*Max_GENERATION*), go to Step 4;
 - Step 12. Output *Best* as the solution.
-

If the computational cost of a node 1 unit time and the number of nodes in an organization is n_{node} , then the whole computational cost of the algorithm is $n_{node} \times M \times MAX\_GENERATION$.

4 Experiments

Table 1 shows the parameters for optimal approximation of linear systems. For stable and unstable linear systems, the maximum number of node in a cell is 20, and for high order and sparse linear systems, it is 30. Only in the experiment of stable linear systems, delay is taken into account and Sample 1 is selected. Sample 2 is selected in the others. Because we have several ways for crossover and mutation, the probability is determined by the way. Each way has a uniform probability to choose.

4.1 Optimal Approximation of Stable Linear Systems

The transfer function^[9] is

$$G(s) = \frac{Y(s)}{U(s)} = \frac{k_d k_{r1} (\tau_{od} s + 1) e^{-\theta_d s}}{(\tau_r s + 1)(\tau_1 s + 1)(\tau_2 s + 1) - k_{r2} k_d (\tau_{od} s + 1) e^{-\theta_d s}}. \quad (3)$$

Table 1. Parameters for optimal approximation of linear systems

Parameter	Values
t_s	5
$MAX\_GENERATION$	200
M	200
Maximum number of nodes in a cell	20 (30)
The frequency	$\omega_i = 10^{-2+\alpha \omega_i}$, $i=0, 1, \dots, N$
Sample 1	$N=50$, $\alpha=0.1$
Sample 2	$N=60$, $\alpha=0.2$

Where $k_{r1}=0.258$, $k_{r2}=0.281$, $k_d=1.4494$, $\theta_d=0.2912$, $\tau_r=1.3498$, $\tau_{od}=0.3684$, $\tau_1=1.9624$, $\tau_2=0.43256$. We set the maximum order 2 for the optimal approximate model. We run the algorithm 20 times independently, and the results indicated that J_s is equal to 10^{-5} in 7 out of 20 times, and is equal to 10^{-6} in 13 out of 20 times. Fig. 5 shows the average and the best fitness over the 20 results.

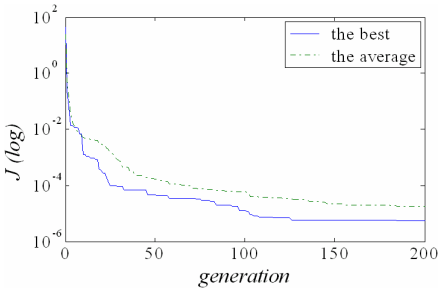


Fig. 5. Convergences of J_s (2-order) on the stable linear system problem

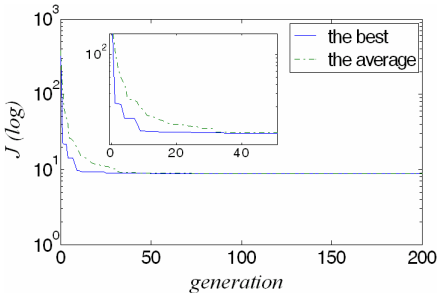


Fig. 6. Convergences of J_s (2-order) on the stable linear system problem

Table 2. Comparisons of the approximate second-order models obtained by MAEA^[8], DEA^[9] and HGP on the stable linear system problem

	Approximate model	J
MAEA	$H_2(s) = \frac{0.0172520(s + 7.7668024)e^{-0.4346119s}}{s^2 + 1.2483717s + 0.2123842}$	2.7180×10^{-5}
DEA	$H_2(s) = \frac{0.0025538(s + 68.1706405)e^{-0.3787929s}}{s^2 + 1.760682s + 0.2759465}$	6.1129×10^{-5}
HGP	$H_2(s) = \frac{0.1359661995e^{-0.316487s}}{s^2 + 1.26469302776s + 0.21555104397}$	5.51149×10^{-6}

The comparisons of the optimal second-order approximate models for the stable linear system are shown in Table 2. The model of MAEA is from [8] and the model of DEA is from [9].

4.2 Optimal Approximation of Instable Linear Systems

The transfer function^[9] is

$$G(s) = \frac{60s^3 + 25850s^2 + 685000s - 2500000}{s^4 + 105s^3 + 10450s^2 + 45000s - 500000}. \quad (4)$$

After generation 50, some J_s are less than 9 and some are around 8.8. They slowly diminish when J_s are less than 8.8. There is a *Best* whose J is 8.79549, and whose ε is 1.3787. $H_2(s)$ of the *Best* is shown in Table 3. The numbers of nodes are 3 and 14 in the numerator and denominator of *Best*, respectively. We run the algorithm 20 times independently, and the results indicated that all J_s are less than 8.9. Fig. 6 shows the average and the best fitness over the 20 results.

4.3 Optimal Approximation of High Order and Sparse Systems

Solving problems of optimal approximation of high order and sparse systems usually needs a large amount of parameters, or else a better model is not obtained with ease. But GP can automatically choose certain valid information and find some better models to get the exact solution. For example:

$$G(s) = \frac{1 + 2s^2 + s^{50} + s^{99}}{1 + 2s + s^{20} + s^{100}}. \quad (5)$$

If we aim to get a 50-order system to approximate (5), it needs 101 parameters with public methods. The problem is complex, but GP does not need all parameters, it can select the right order to approximate systems automatically. In order to describe the sparseness of linear systems, a parameter, namely sparse degree of linear systems is defined as follows:

Definition 1: Sparse Degree of Polynomial, which is labeled as ρ^{poly} , is defined as the ratio of the number of coefficient to the highest order adding 1 in the polynomial.

Definition 2: Let the sparse degree of polynomial for the polynomials in the numerator and the denominator of a linear system be ρ_n^{poly} and ρ_d^{poly} , respectively. Thus, Sparse Degree of Linear Systems, which is labeled as ρ^s , is defined as $\max(\rho_n^{poly}, \rho_d^{poly})$.

For example, in (5), $\rho_n^{poly} = 4/100 = 0.04$, $\rho_d^{poly} = 4/101 = 0.0396$, and $\rho^s = 0.04$.

When the maximum order is 50, the maximum number of nodes in a cell is 30, and the max exponent of the terminal s is 40, an optimal approximate model is obtained at generation 10, i.e.

$$H(s) = \frac{4.08737 - 0.055s}{13.686624s + 4.203404}$$

Where the numbers of nodes in the numerator and denominator are 3 and 4, respectively. $J=1.13746$, $\varepsilon=0.779071$. Change the maximum number of nodes in a cell to 15, another optimal approximate model is obtained at generation 50, i.e.

$$H(s) = \frac{0.89 + s^2 + s^{16} + 2s^{19} - s^{21} - s^{30}}{0.904 + 2.18578s + 4.18578s^{10} + s^{25} - s^{31}}$$

Where the numbers of nodes in the numerator and denominator are 8 and 9, respectively. $J=0.514806$, $\varepsilon=0.177624$. We run the algorithm 10 times independently, and all J_s are less than 0.1.

For another system:

$$G(s) = \frac{100 + 3.0s^2 + 5.3s^{15} + 3.3s^{20} + 0.3s^{40} + 0.3s^{50} + 0.2s^{60} + 0.1s^{70} + s^{90}}{1 + 1.2s + 5.2s^{60} + s^{100}}. \quad (6)$$

where $\rho_n^{poly} = 0.0989$, $\rho_d^{poly} = 0.0396$. We run the algorithm 10 times independently, and all J_s are less than 1. One optimal approximate model is

$$H(s) = \frac{350.277 + 2s + s^2 + s^9 + 4.78818s^{13} + 254.332161s^{14} + 8.867s^{17} + 8.867s^{21} - s^{25}}{3.505 + 4.250s - s^{10} + s^{19} - s^{26} - s^{30}}$$

Where the number of nodes in the numerator and denominator are 16 and 6, respectively. $J=0.225249$, $\varepsilon=0.283741$.

4.4 Summary of the Experimental Results

Table 2 shows that the number of parameters of the optimal model obtained by HGP is less than those of MAEA and DEA. Table 3 shows that ε of HGP is less than those of MAEA and DEA. Even though the approximate model is unknown, HGP can create the optimal approximate model automatically. Through the optimal approximation of high order and sparse linear systems, it is known that HGP can choose valid information automatically. Tables 2 and 3 also show that J of HGP is better than those of MAEA and DEA.

To summarize, HGP can perform well for optimal approximation of linear systems even though in the case that the approximate model and the range of the parameters are unknown. HGP can be easily used to search an optimal approximation of high order and sparse linear systems. It is of great reference value to the aviation industry and other applied fields.

Table 3. Comparison of the approximate second-order models obtained by MAEA^[8], DEA^[9] and HGP on the instable linear system problem

	Approximate model	J	ε
MAEA	$H_2(s) = \frac{129.315s-451.012}{s^2+13.092s-91.812}$	8.79546	1.38002
DEA	$H_2(s) = \frac{129.297s-450.711}{s^2+13.089s-91.7495}$	8.795	1.449
HGP	$H_2(s) = \frac{129.4261s-451.3264}{s^2+13.1016s-91.8666}$	8.79549	1.3787
DEA	$H_3(s) = \frac{198.577s^2+2561.64s-10871}{s^3+52.786s^2+104.519s-2168.14}$	3.160	1.117
HGP	$H_3(s) = \frac{198.4614s^2+2604.13s-11236.76}{s^3+52.06157s^2+104.0798s-2241.419}$	3.1632	1.0960

5 Conclusions

HGP proposed in this paper has fewer introducing parameters, and performs robust. HGP can automatically search a global optimal solution and does not need prior knowledge. Study operator is taken into account, which could fasten the convergence rate. Thus, it can be flexible to deal with approximation for optimal high order and sparse systems. Our future works will apply HGP to other problems, such as the artificial ant and even-n-parity problems.

References

1. Hollabd, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Application to Biology, Control, and Artificial Intelligence, 2nd edn. MIT Press, Cambridge (1992)
2. Koza, J. R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
3. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: Generic Programming: An Introduction on the Automatic Evolution of Computer Programs and its Applications. Morgan Kaufmann, San Francisco (1998)
4. Oltean, M.: Multi-expression Programming, Technical Report, Babes-Bolyai Univ, Romania (2006)
5. Ferreira, C.: Gene Expression Programming: A New Adaptive Algorithm for Solving Problems. Complex Systems 13, 19–87 (2001)
6. Miller, J.F., Job, D., Thomson, P.: Cartesian genetic programming. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) EuroGP 2000. LNCS, vol. 1802, pp. 131–132. Springer, Heidelberg (2000)
7. Fortuna, L., Nunnari, G., Gallo, A.: Model Order Reduction with Applications in Electrical Engineering. Springer, London (1992)

8. Zhong, W., Liu, J., Xue, M., Jiao, L.: A multiagent genetic algorithm for global numerical optimization. *IEEE Trans. System, Man, and Cybernetics—Part B* 34(2), 1128–1141 (2004)
9. Cheng, S., Hwang, C.: Optimal approximation of linear systems by a differential evolution algorithm. *IEEE Trans. Syst., Man, Cyber. A* 31(6), 698–707 (2001)
10. Guo, T.Y., Hwang, C.: Optimal reduced-order models for unstable and nonminimum-phase systems. *IEEE Trans. Circuits and Systems I* 43(9), 800–805 (1996)
11. Liu, J., Zhong, W., Jiao, L.: An organizational evolutionary algorithm for numerical optimization. *IEEE Trans. System, Man, and Cybernetics—Part B* 37(4), 1052–1064 (2007)
12. Wang, G., Soule, T.: How to Choose Appropriate Function Sets for Genetic Programming. In: Keijzer, M., O'Reilly, U.-M., Lucas, S., Costa, E., Soule, T. (eds.) *EuroGP 2004*. LNCS, vol. 3003, pp. 198–207. Springer, Heidelberg (2004)

Genetic Vector Quantizer Design on Reconfigurable Hardware

Ting-Kuan Lin¹, Hui-Ya Li¹, Wen-Jyi Hwang^{1,*}, Chien-Min Ou²,
and Sheng-Kai Weng¹

¹ Department of Computer Science and Information Engineering,
National Taiwan Normal University, Taipei, 117, Taiwan
{g95470270,g96470019,whwang,g96470502}@csie.ntnu.edu.tw

² Department of Electronics Engineering, Ching-Yun University,
Chungli, 320, Taiwan
cmou@cyu.edu.tw

Abstract. This paper presents a novel hardware architecture for genetic vector quantizer (VQ) design. The architecture is based on steady-state genetic algorithm (GA). It adopts a novel architecture based on shift registers for accelerating mutation and crossover operations while reducing area cost. It also uses a pipeline architecture for fitness evaluation. The proposed architecture has been embedded in a softcore CPU for physical performance measurement. Experimental results show that the proposed architecture is an effective alternative for VQ optimization attaining both high performance and low computational time.

1 Introduction

Genetic algorithm (GA) [2,7] is a general-purpose search algorithm for solving optimization problems by simulating natural evolution over populations of candidate solutions. Inspired by biological evolution, the GA consists of a set of genetic strings, which are evaluated by a fitness function. The fittest strings are then regenerated at the expense of the others. Moreover, crossover and mutation are employed to obtain better strings. The mutation operator changes individual elements of a string, and the crossover operation interchanges parts between strings. The combination of these operations is called a generation. The evolution of genetic strings may be continued for several generations to attain a near global optimal solution.

One drawback of GA is its computational complexity. When the population size for genetic operations is large, the computational time for some GA applications may be high. Nevertheless, it is not favorable to reduce population size because small population size will trap the GA in poor local optimum. One way to lower the execution time of GAs is to implement the algorithms in hardware [1,8,10]. However, existing GA hardware implementations have a number of drawbacks. First of all, large storage size is required for processing the genetic

* To whom all correspondence should be sent.

strings. Usually two set of population memories are used for the regeneration process. One memory contains the parent strings; the other stores the child strings after the regeneration. In addition, there is overhead for switching one memory to another at the beginning of a new generation. The second drawback is that the regeneration process is based on the fitness function. The selection of parents therefore may need large chip area for hardware implementation. In addition, the mutation and crossover operations also result in high area cost when concurrent processing over all the genetic strings is desired.

The objective of this paper is to present a novel hardware architecture for fast GA optimization eliminating the drawbacks stated above. In this paper, we consider the applications of GA for vector quantizer (VQ) [3] design. When applied for VQ training, the GA requires large storage size and long training time [4]. Therefore, the VQ design is a good example for verifying the effectiveness of the proposed GA architecture. The design consists of population memory unit, mutation and crossover unit, fitness evaluation unit, and survival test and update unit. It contains only one population memory for reducing the area cost. Both the mutation and crossover operations are performed concurrently for accelerating the GA. In addition, a pipeline architecture with direct memory access (DMA) operation is adopted for the fitness function evaluation. A hardware sorting structure is adopted for survival test. The proposed architecture has been implemented on field programmable gate array (FPGA) devices [6] so that it can operate in conjunction with a softcore CPU [12]. Using the reconfigurable hardware, we are then able to construct a system on programmable chip (SOPC) system for the genetic VQ design. As compared with its software counterparts, numerical results reveal that the proposed FPGA-based GA architecture attains higher performance with significantly lower training time for VQ design. These facts demonstrate the effectiveness of our design.

2 Preliminaries

The goal of a VQ for data clustering is to partition a large data set $\mathcal{X} = \{x_1, \dots, x_t\}$ into N non-overlapping clusters $\mathcal{C}_1, \dots, \mathcal{C}_N$, where $N \ll t$. The partitioning process is based on a set of codewords $\{y_1, \dots, y_N\}$, where the codewords and the vectors in $\mathcal{X}$ are of the same dimension w . Given a vector $x \in \mathcal{X}$, the x will be assigned to the cluster $\mathcal{C}_i$ when

$$i = \alpha(x) = \arg \min_{1 \leq j \leq N} d(x, y_j), \quad (1)$$

where $d(u, v)$ denotes a distance measure between two vectors u and v . In this paper, the squared distance is adopted as the distance measure. When applied for data reduction applications such as data compression, a vector x will be represented by the codeword y_i when $i = \alpha(x)$. One cost function for the data reduction is the average distortion for representing x by y_i , as shown below

$$D = \frac{1}{wt} \sum_{i=1}^t d(x_i, y_{\alpha(x_i)}). \quad (2)$$

Given a data set $\mathcal{X}$, the objective of the VQ design is to find a set of codewords $\{y_1, \dots, y_N\}$ minimizing D in eq.(2).

In the basic GA (termed GA (I)) for VQ design, there are P genetic strings for the genetic operations. Each string r represents a set of N codewords $\{y_1, \dots, y_N\}_r$. Note that these strings are strings of vectors, not strings of binary numbers.

Let $S(k)$ and $D(k)$ denote the set of P strings and the value of current minimum distortion D after the execution of the k -th generation of the basic GA, respectively. Let s^* be the current optimum string during the course of genetic operations. Suppose that the $(k-1)$ -th iteration is completed, and the execution of the k -th ($k \geq 1$) is to be done. We then perform the following genetic operations sequentially on the strings in $S(k-1)$.

Regeneration: Since each string in $S(k-1)$ for the genetic operations is in fact a codebook of VQ, its corresponding D can be computed using eq.(2). The inverse of D is used as fitness function for each string. The regeneration process is then conducted using the roulette-wheel technique. Once a string has been selected for reproduction, an exact replica of it is made as a regeneration string. In the algorithm, P regeneration strings are created after the regeneration operation.

Crossover: On each regeneration string r , $\{y_1, y_2, \dots, y_N\}_r$, one point crossover is applied with probability P_c . Out of the total population, a partner string r' , $\{z_1, z_2, \dots, z_N\}_{r'}$, is randomly chosen. Then an integer random number n between 1 and N is generated. Both strings are cut into two portions at position n and the portions $\{y_{n+1}, \dots, y_N\}$ and $\{z_{n+1}, \dots, z_N\}$ are mutually exchanged.

Mutation: Mutation is performed on each codeword of each string with a small probability P_m . Suppose now the string $r = \{y_1, y_2, \dots, y_N\}_r$ is to be mutated. One of the N codewords, y , is chosen at random. Among the w numbers in y , we also select one number at random. Then a random number, taking the binary values b or $-b$, is generated, and is added to the chosen component.

In the GA algorithm, the iteration continues until the convergence of the sequence $D(k)$. The current optimum string s^* after the completion of GA algorithm is then chosen as the desired result.

It may be difficult to implement the GA (I) algorithm in hardware. This is because two population memories are required in the algorithm. One population memory contains the parent strings. The other is used for storing the strings after the regeneration process. The roulette-wheel technique may also become the bottleneck for the hardware. In addition, the crossover and mutation operations algorithm should operate over all the P regenerated strings. The corresponding hardware complexity therefore may be very high.

3 The Proposed GA Architecture

In this paper, an alternative GA algorithm (termed GA (II) algorithm) is adopted for the VQ design. The GA (II) is based on the steady-state GA, which has superior performance over the basic generational GA for a number of

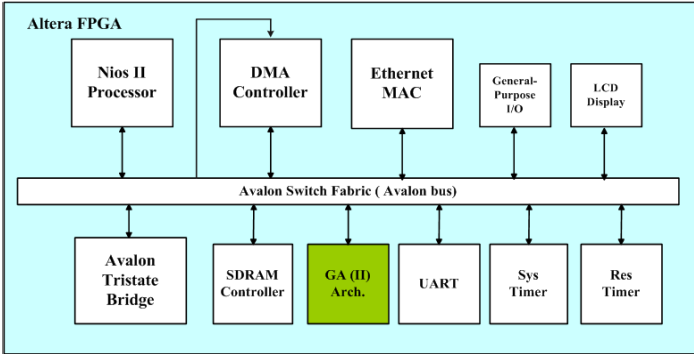


Fig. 1. The employment of SOPC for the GA (II) architecture

applications [9]. There is no concept of generation in steady-state GA. Let S be the set of P genetic strings, which are called the parent strings. Initially, the P strings in S are randomly generated. Two strings (denoted by r_1 and r_2) in S will be selected for mutation and crossover for creating a new child string (denoted by c). The fitness value of the child string is then evaluated and compared with the fitness value of all the parent strings in S . If the new string is inferior to all the parent strings in S , no parent string will be removed. Otherwise, the parent string with lowest fitness value is replaced by the child string.

Note that because each string for the VQ design is actually a codebook, the memory access time for string retrieval may be long. Consequently, the retrieval process for r_1 and r_2 may be time-consuming. To reduce the memory access time, in the algorithm, the previous r_1 becomes the new r_2 and then the new r_1 is chosen randomly from S . This selection scheme reduces the memory access time by half.

As the process of selection, crossover, mutation, and survival/replacement continues, the overall fitness of population will increase and the survival rate of new off-spring will diminish. At some point, the offspring survival rate will drop to zero. At this point, evolution has probably ceased and the algorithm may be terminated. The GA (II) algorithm is more effective for the hardware design. Only one population memory is required. In addition, crossover and mutation operations only operate on r_1 and r_2 instead of all strings in the population memory.

The hardware architecture of the GA (II) can be viewed as a user logic in the NIOS-based SOPC system [12], as shown in Fig. 1. Because the proposed architecture is used for the VQ design, the training data is required. The goal of using the SOPC is to provide the training data for the hardware architecture. The training data can be stored in a SDRAM, and delivered to our architecture via the Avalon bus. The DMA can be used to speed up the delivery. Alternatively, the training data can be obtained from a remote host via the internet.

Fig. 2 shows the hardware architecture of the GA (II) algorithm. It contains population memory, crossover & mutation unit, fitness evaluation unit, survival

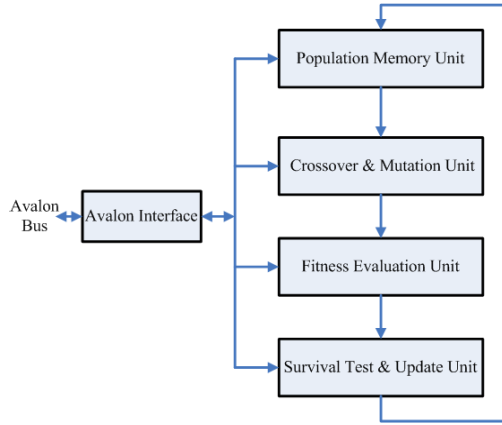


Fig. 2. The proposed hardware architecture for GA (II)

test & update unit, and Avalon bus interface. Both the population memory and crossover & mutation unit contain random number generators (RNGs).

In this architecture, the population memory unit is devoted for storing the genetic strings. Moreover, the random selection of parent strings for subsequent crossover and mutation operations is also included here. This selection is based on the RNG inside the population memory unit. All the crossover and mutation operations are performed concurrently in the crossover & mutation unit for producing a new child string c . The fitness value of the resulting string is then evaluated by the fitness evaluation unit.

Based on the fitness value, the goal of the survival test & update unit is to determine whether the child string c will survive. If it is the case, the parent string in the population memory unit with the worst fitness value will be replaced by the child string. Each unit in Fig. 2 will be described in detailed as shown below.

Population Memory Unit: The population memory contains a 2-port RAM and a RNG unit. The 2-port RAM contains S , the set of P genetic strings. In our design, the implementation of the RAM is based on the embedded memory, which is provided by some FPGA devices such as Altera Stratix II. The goal of RNG unit in the population memory unit is to select randomly a string r_1 for the subsequent crossover and mutation operations. In our design, the cellular automata (CA) is adopted for the VLSI implementation of random number generator due to its simplicity and regularity of the design.

Mutation and Crossover Unit: Fig. 3 shows the basic structure of the mutation and crossover unit, which contains three shift registers for storing the strings r_1 , r_2 and c , respectively. A number of RNGs, comparators, multiplexers and counters are then used for crossover and mutation. The major advantage of this architecture is that the crossover and mutation can be performed concurrently with low area cost.

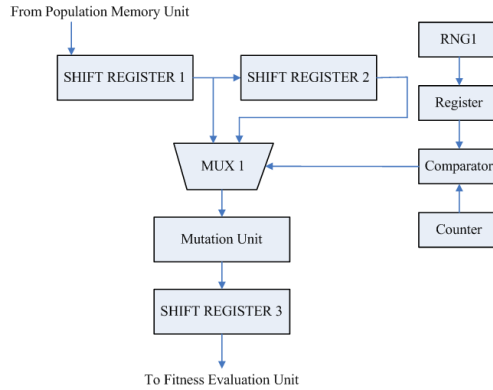


Fig. 3. The architecture of crossover and mutation unit

As shown in Fig. 3, SHIFT REGISTER 1 and SHIFT REGISTER 2 contain strings r_1 and r_2 , respectively. Note that the architecture does not randomly select new r_1 and r_2 from the population memory. In fact, only new r_1 is chosen from population memory. The new r_2 is actually the previous r_1 . The memory access time and routing overhead can then be significantly reduced. Based on the algorithm, in the architecture, The SHIFT REGISTER 1 obtains r_1 from the population memory unit. The SHIFT REGISTER 2 obtains r_2 from SHIFT REGISTER 1.

The crossover operations are accomplished by concurrently shifting the strings in SHIFT REGISTER 1 and SHIFT REGISTER 2 to MUX 1. Each shift register will shift one codeword at a time. As shown in Fig. 3, MUX 1 is a switch selecting the codewords of either r_1 or r_2 , and route them to SHIFT REGISTER 3, which contains the resulting child string c . The control line of MUX 1 is connected to a comparator, which compares the value of RNG 1 to that of a counter. The counter records the number of shifts made by the shift registers. The value of RNG 1 serves as a threshold here. When the counter value is less than the threshold, the codeword of SHIFT REGISTER 1 (i.e., r_1) goes to SHIFT REGISTER 3. Otherwise, codewords of r_2 will be selected. Consequently, the value of RNG 1 determines the crossover point. The value will be randomly generated prior to the shifting operations.

We also observe from Fig. 3 that the output codeword of MUX 1 will pass through the mutation unit before arriving the SHIFT REGISTER 3. Fig. 4 shows the architecture of the mutation unit. As shown in the figure, all w components of the output codeword mutate concurrently. The mutation circuit for each component i consists of 2 RNGs (termed RNG $_{ia}$ and RNG $_{ib}$), one register (termed register i), one comparator (termed comparator i), and one multiplexer (termed mux i).

The probability for mutation P_b is stored in a separate register, and is broadcasted to all the mutation circuits. In the mutation circuit for each component i , the value of RNG $_{ia}$ is first compared with the P_b . The component i will be

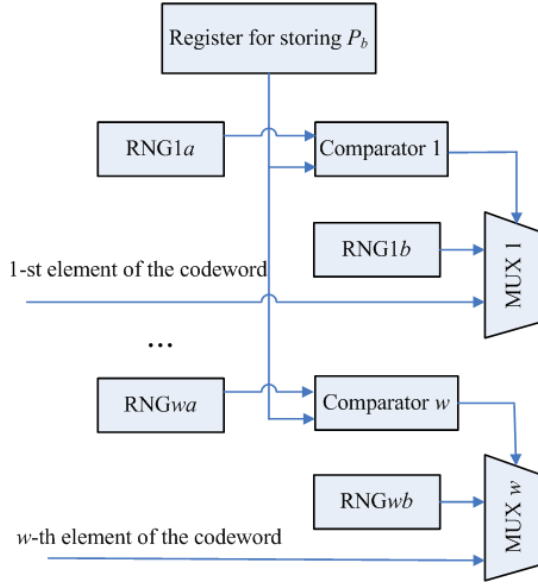


Fig. 4. The architecture of mutation unit

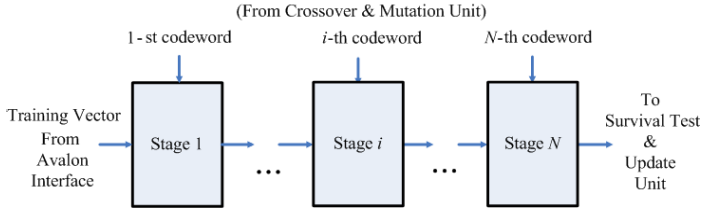


Fig. 5. The architecture of the fitness evaluation unit

mutated when the value of RNG_{ia} is less than P_b . The mutated value is then determined by RNG_{ib} .

Fitness Evaluation Unit: The goal of the fitness evaluation unit is to compute the average distortion of the mutated child string stored in SHIFT REGISTER 3 using eq.(2). Fig. 5 shows the architecture of the fitness evaluation unit, which is an N -stage pipeline, where N is the number of codewords. The pipeline fetch one training vector $x \in \mathcal{X}$ per clock from Avalon bus using the DMA operation, where the vectors in the training set $\mathcal{X}$ are stored in the SDRAM shown in Fig. 2.

The i -th stage, $i = 1, \dots, N$ of the pipeline compute the squared distance between the training vector at that stage and the i -th codeword of the child string stored in the SHIFT REGISTER 3 of the mutation and crossover unit shown in Fig. 3. The squared distance is then compared with the current minimum distance up to the i -th stage. If distance is smaller than the current minimum,

Table 1. Area cost and operating speed of the SOPC system using the proposed GA architecture as custom user logic

Items	Values
ALMs of GA (II) Circuit	6475
Embedded Memory Bits of GA (II) Circuit	131072
DSP 9-bit Elements of GA (II) Circuit	256
ALMs of Entire SOPC System	12126
Embedded Memory Bits of Entire SOPC System	727808
DSP 9-bit Elements of Entire SOPC System	264
Operating Frequency	50MHz

then the i -th codeword becomes the new current optimal codeword, and the corresponding distance becomes the new current minimum distance. After the computation at the N -th stage is completed, the current optimal codeword and current minimum distance are the actual optimal codeword and the actual minimum distance, respectively.

Survival Test and Update Unit: This unit contains a hardware sorting circuit [5], which sorts the N parent strings in a descending order according to their fitness values. After the fitness evaluation operation is completed, the fitness value of the child string is used as the input to the sorting circuit. When the distortion of the string is larger than the parent string with lowest fitness value, the child string is not survival, and no updating operation is necessary. Otherwise, the parent string with highest distortion is replaced by the child string. The sorting circuit is then activated to determine the new parent string with the highest distortion.

4 Experimental Results and Concluding Remarks

This section presents some physical performance measurements of the proposed FPGA implementation. In the experiments, the number of genetic strings $P = 64$. The vector dimension of codewords is $w = 2 \times 2$. There are 64 codewords (i.e., $N = 64$) in the VQ. The mutation probability is $P_b = 0.03125$. The target FPGA device for the hardware design is Altera Stratix II 2S60 [11].

Table 1 shows the area cost and the operating speed of the NIOS-based SOPC system [12] using the proposed GA architecture as the custom user logic, as shown in Fig. 1. The proposed GA circuit consumes only 6475 adaptive logic modules (ALMs) [11]. The population memory of the architecture is implemented by the embedded memory of the FPGA. The consumption of the population memory bits for the GA (II) circuit is 131072 bits. Moreover, our circuit also uses 256 digital signal processing (DSP) 9-bit elements of the FPGA device for the implementation of squared distance computation in the fitness evaluation unit. The NIOS softcore CPU [12] itself also consumes hardware resources. However, the whole SOPC system uses 12126 ALMs, which is only 50 % of the ALMs of the target FPGA device. The operating speed of the system is 50 MHz.

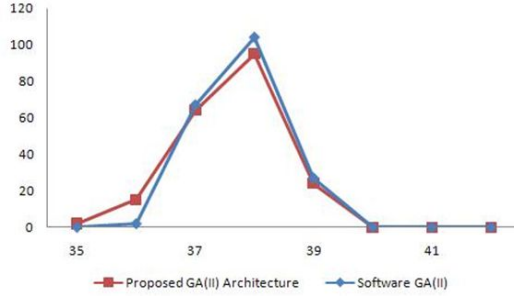


Fig. 6. Distortion Distribution of proposed GA(II) architecture and its software counterpart

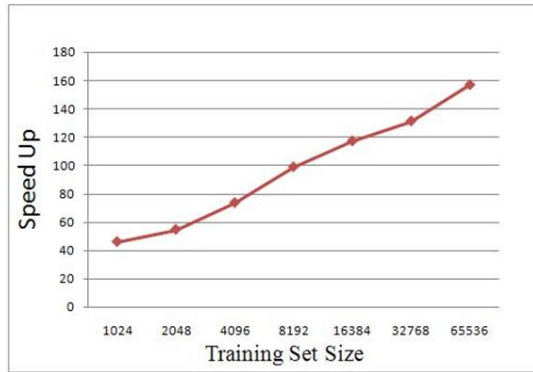


Fig. 7. Speedup of the proposed GA architecture over its software counterpart for various training set sizes

Fig. 6 compares the average distortion of the proposed FPGA GA (II) implementation with that of its software counterpart under the same number of genetic string $P = 64$, number of codewords $N = 64$, and vector dimension $w = 2 \times 2$. The software implementation of GA (II) is executed on the 3-GHz Pentium IV CPU. In the experiment, we execute each implementation 200 times independently. The training set contains 65536 training vectors from the 512×512 image “Lena.” Based on the training set, the average distortion of each execution is computed according to eq.(2). Fig. 6 then shows the distribution of the average distortion.

Note that, in the software GA (II) implementation, new strings r_1 and r_2 are randomly selected from the population memory S for crossover and mutation. Nevertheless, for our hardware GA (II) implementation, only the new r_1 is chosen randomly from S . The new r_2 is actually the old r_1 for reducing the memory access time. It can be observed from Fig. 6 that both implementations have similar average distortion distributions. Therefore, our architecture simplifying the string random selection process does not degrade the performance of genetic VQ design.

Fig. 7 shows the speedup of the proposed hardware architecture over its software counterpart for various training set sizes. The speedup is defined as the execution time (3-GHz Pentium IV CPU) of the software implementation divided by the execution time (50-MHz NIOS softcore CPU) of the SOPC system using the proposed GA architecture as the custom user logic. As shown in the figure, the speedup increases with the training set size. This is because the training set is used for the fitness evaluation in the GA. The training set is often stored in the main memory, and therefore requires long memory access time. In addition, the inverse of the average distortion D in eq.(2) is used as the fitness function. The computational complexity therefore is high. In our architecture, the DMA and pipeline techniques are used for reducing the memory access time and the computational time for fitness evaluation. Therefore, our design has high speedup over the software implementation when the training set size is large. In particular, when the training set contains 65536 vectors, the execution time of our SOPC system is 329ms; whereas, the execution time of software system is 51776 ms. The speedup is 157. These facts demonstrate the effectiveness of our design.

References

1. Choi, Y.H., Chung, D.J.: VLSI Processor of Parallel Genetic Algorithm. In: IEEE Asia Pacific Conf. on ASICs, pp. 143–146 (2000)
2. Eiben, A.E., Smith, J.D.: Introduction to Evolutionary Computing. Springer, Heidelberg (2003)
3. Gersho, A., Gray, R.M.: Vector Quantization and Signal Compression. Kluwer, Norwood (1992)
4. Hwang, W.J., Hong, S.L.: Genetic entropy-constrained vector quantization. Optical Engineering 38, 233–239 (1999)
5. Hwang, W.J., Li, H.Y., Yeh, Y.J., Chan, K.F.: FPGA Implementation of Competitive Learning with Partial Distance Search in the Wavelet Domain. In: Kang, G.B. (ed.) Progress in Neurocomputing Research, ch. 8, pp. 203–221. NOVA Science Publisher (2008)
6. Hauck, S., Dehon, A.: Reconfigurable Computing. Morgan Kaufmann, San Francisco (2008)
7. Mitchell, M.: An introduction to genetic algorithm. MIT Press, Cambridge (1996)
8. Nedjah, N., Mourelle, L.: Hardware Architecture for Genetic Algorithms. In: Ali, M., Esposito, F. (eds.) IEA/AIE 2005. LNCS, vol. 3533, pp. 554–556. Springer, Heidelberg (2005)
9. Rasheed, K., Davisson, B.D.: Effect of global parallelism on the behave of a steady state genetic algorithm for design optimization. In: Proceedings of the Congress on Evolutionary Computation, Washington, DC (1999)
10. Tommiska, M., Vuori, J.: Implementation of genetic algorithms with programmable logic devices. In: Proc. 2nd Nordic Workshop on Genetic Algorithms and Their Applications, pp. 111–126 (1996)
11. Stratix II Device Handbook, Altera Corporation (2008), <http://www.altera.com/literature/lit-nio2.jsp>
12. NIOS II Processor Reference Handbook, Altera Corporation (2008), <http://www.altera.com/literature/lit-nio2.jsp>

Pattern Learning and Decision Making in a Photovoltaic System

Rongxin Li and Peter Wang

Autonomous Systems Laboratory, CSIRO ICT Centre, Australia
`{Ron.Li,Peter.Wang}@csiro.au`

Abstract. We study the effects of different decision making schemes on the accumulative rewards when photovoltaic (PV) facilities are intended as a potential replacement for conventional peaking power plants. As the amount of solar irradiance usable by a PV module follows a stochastic process, we compare the outcomes using the following two strategies in a stochastic environment: (1) employing an optimal decision making approach without any specific knowledge of the environment; and (2) optimal decision making based upon learning patterns of the environment process. We examine the possibility of integrating a pattern learning approach – called an ϵ -Machine – with a Partially Observable Markov Decision Process (POMDP). This approach has been motivated in part by the fact that efforts in extending traditional learning approaches to POMDPs have so far achieved only limited success. The PV facility in our model consists of a PV panel and a battery, with an associated local, non-critical load. Under the assumption that any PV generated power exceeding the maximum local consumption capacity must be dumped when the battery is full, the goal of the autonomous control agent is to maintain the maximum output potential to most effectively offset unexpected demand peaks, while minimizing energy wastage in the presence of strong solar irradiance. The environment is assumed to follow a Markov process of a different order than the part of the system under the influence of the agent.

1 Introduction

Peak electricity demand in Australia has been increasing at a faster rate than the base load demand. Conventional peaking power plants, however, are not cost-effective to build and maintain, as they are either idle or on stand-by most of the time. These generators also produce a significant amount of greenhouse gases. Reducing the need for such generators therefore has potentially significant socio-economic benefits.

Photovoltaic (PV) generation of electricity constitutes an important renewable energy source. However, obstacles to its widespread use include its intermittency, volatility and unpredictability. The scheme that we examine in this paper uses solar power for local non-critical use (e.g. hot water tanks, pool pumps) most of the time, but maintains the storage reserve at or near the maximum

capacity in anticipation for unexpected demand peaks. An alternate possibility is to inject the electricity to the power grid whenever possible while maintaining the storage reserve for offsetting demand peaks; however this possibility is not explicitly discussed in this paper.

A second consideration is to avoid energy wastage. Given that any power exceeding the maximum local consumption ability must be dumped when the battery is full, wastage can happen in the presence of strong irradiance. Our goal is to maintain the maximum capacity to offset unexpected demand peaks, while minimizing any wastage. This requires optimal decision making within a stochastic environment.

As a decision-theoretic approach, Markov decision processes (MDPs) mathematically model planning problems where outcomes are partly under the control of the decision maker despite some uncontrollable randomness [1,2]. The MDP provides a powerful theoretical framework in which sequential decision making in probabilistic environments can be formalized. The purpose of employing such a framework is usually to enable decisions to be made that lead to an optimal cumulative measure. In some applications, however, an agent may be unable to acquire precise knowledge of the true state of the global environment. Therefore, an extended MDP model referred to as the Partially Observable Markov Decision Process (POMDP) may be more applicable, where decision making is based upon a probabilistic distribution over environment states inferred by an autonomous control agent from its partial or unreliable observations. Such a probabilistic distribution is referred to as a belief state. The policy for a POMDP prescribes the best action to be taken for any given belief state.

Our work is motivated in part by the fact that efforts in extending traditional learning approaches (such as reinforcement learning) to POMDPs have so far achieved only limited success. We investigate the possibility of integrating a pattern-learning approach, referred to as the ϵ -Machine [3], with POMDPs. The environment is assumed to possibly follow a Markov process of a different order (i.e. memory length) than the part of the system subject to the influence of the agent. A discretized irradiance process is assumed to follow a Markov process of order L_{max} . In essence, an ϵ -Machine is a model that minimally describes the patterns in a stochastic process with the least number of hidden Markovian states. These states are statistically capable of producing the behavior exhibited in the data, and are thus called the underlying process's causal states [3]. Based on historical data the ϵ -Machine learns all possible causal states up to a size of L_{max} . The required transition probabilities are then computed based on the ϵ -Machine's knowledge. These probabilities may be regularly updated to ensure the optimal policy resultant from solving the POMDP stays current.

2 Background

2.1 Mathematical Formulations of MDP and POMDP

As a stochastic control process, an MDP is defined by a tuple S, A, T, R , where S is the finite set of environment states; A is the set of possible actions that may

be taken by the decision maker, $T : S \times A \times S \rightarrow [0, 1]$ is the probabilistic state-transition model, and $R : S \times A \rightarrow \mathbb{R}$ is the immediate reward or cost expected to be received after a transition. An underlying assumption is that the process is stationary and Markovian. The goal is to maximize or minimize an expected cumulative measure, usually a discounted sum of the immediate rewards or cost. The policy $\pi(s)$ for an MDP may be computed via algorithms such as value iteration or policy iteration, which prescribes the best action to be taken in a given state $s \in S$.

With an additional finite set of observations O , a POMDP is defined as the tuple (S, A, T, R, O) . POMDPs can be converted to an MDP defined over a continuous space of belief states. Suppose the agent believes that the current likelihood function over all possible states $\mathbf{s}$ is $b_t(s)$. Upon observing $o \in O$ after taking action $a \in A$, which is known to have the probability $p(o \mid s', a)$ if the environment is in state s' , the agent should update its belief about the current likelihood of s' .

Further extensions may be necessary in more restrictive situations, such as when the local reward is received by each agent. Decentralized formulations of MDP or POMDP are more appropriate in such scenarios, where joint rewards must also be deduced from a statistical model. In this paper, however, we focus only on the local decision making aspects of the control agents.

2.2 The ϵ Machine

An ϵ -Machine is a minimal stochastic sequence generator based on learning patterns in an observed sequence. Formally, we consider a discrete-time, discrete-valued Markovian process, represented by the bi-infinite sequence $\vec{S} = \cdots S_{t-1} S_t S_{t+1} \cdots$, where S_t is a random variable sampled at time t . $\vec{S}$ is split into a past series $\bar{S} = \cdots S_{t-2} S_{t-1}$ and a future sequence $\vec{S} = S_t S_{t+1} \cdots$. The conditional probability distribution of a future sequence is $P(\vec{S} \mid \bar{S} = \bar{s})$, where $\bar{s}$ is a past series of observations.

The ϵ -Machine for $\vec{S}$ is defined as an ordered pair $\{\epsilon, \mathbf{T}\}$, where ϵ is the following function that maps from histories to sets of histories [4]:

$$\epsilon(\bar{s}) \equiv \{\bar{s}' \mid P(\vec{S} \in F \mid \bar{S} = \bar{s}) = P(\vec{S} \in F \mid \bar{S} = \bar{s}'), \forall F \in \mu, \bar{s}' \in \bar{S}\}, \quad (1)$$

where μ is the collection of all measurable future events, and $\mathbf{T}$ is the set of the transition matrices for the states defined by ϵ .

$\epsilon(\cdot)$ defines sets of equivalent historical patterns. In practice, instead of bi-infinite sequences, a maximum length L_{max} is usually defined to restrict the lengths of $\bar{S}$ and $\vec{S}$. More details of building and applying an ϵ -Machine are provided in Section 3.2.

An important concept is the recurrent state. A recurrent state is a causal state, where for each transition to another causal state, there exists a series of transitions that will result in returning to the original causal state. An ϵ -Machine can essentially be described as a deterministic finite state machine that consists of only recurrent states and the associated transitions. The recurrence

constraint ensures that only valid patterns in the observed sequence are learned by the ϵ -Machine.

3 Adaptive Decision Making in a Stochastic Environment

3.1 Observations and the State Space

Charges from the solar panel can be precisely measured by a current meter. Via voltmeters, observations are made of the likely charge status of the battery. The state space in our problem is the product of the battery charge state and the photovoltaic output. This space is discretized. One of the dimensions of the state space, the output of a solar module is fully observable. On the other dimension, the partial observability arises not from unknown state features, but from the extent to which the charge status of a battery can be determined from a voltmeter reading. Thus, the partial observability may be modeled with a bounded noise model.

3.2 Learning Patterns in the PV Process

The main component of a photovoltaic (PV) system is an array of solar cells made from semi-conducting material that generates current when exposed to sunlight by converting optical energy to electrical energy. A PV system also contains a charge controller that controls the rate of battery charging. An inverter is used in a grid-connected system to convert direct current to alternating current.

The maximum power of a solar cell is nearly proportional to the irradiance and decreases with rising temperature. Based on its operating properties, a raw PV module can be idealized as a current source. That is, when operating below a certain voltage it outputs an almost constant current regardless of the impedance of the load. The impedance of the load dictates the output voltage of the PV module within this range. There exists a point on the current-voltage characteristic curve of the PV module, where, soon after the onset of the current's tendency to start decreasing with voltage, the product of current and voltage reaches the maximum [5]. A PV's maximum power point (MPP) changes with the irradiance level and the surface temperature. A maximum power point tracker (MPPT) can be used to track the PV module's MPPT by changing its operation conditions. When a battery is connected to the MPPT, the output voltage is dictated by the charge controller. Thus, the PV composite module - inclusive of the MPPT and the charge controller - effectively becomes a voltage source.

The output current of an MPPT can be accurately measured. Using historical data logs, instantaneous readings at irregular intervals of the output of a PV panel are integrated and re-sampled at regular time intervals. This converts the current (in Amperes) to charge amounts (in Coulombs). A sample output of an MPPT over several days is displayed in Figure 1. These are deemed to be amounts available for immediate local consumption or for battery charging. These continuous amounts are discretized into non-uniform bins in such a way

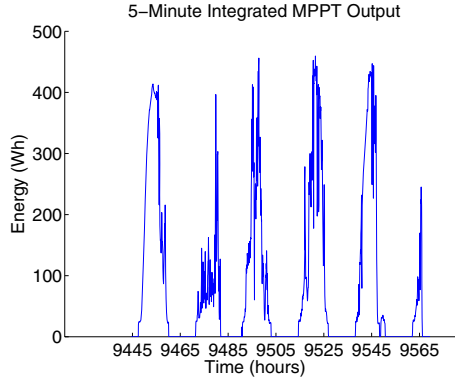


Fig. 1. Sample photovoltaic output over 5-min intervals spanning several days

that the number of data points falling into each bin is as constant as possible. This discretization method is often referred to as adaptive quantization.

We then split the discretized data into a historical set and a test set. From the historical set, statistical patterns are then learned with the ϵ -Machine approach. An algorithm called Causal State Splitting Reconstruction (CSSR) has been proposed to approximate the ϵ -Machine [3]. We employ a slight variation of the CSSR, which can be summarized in the following four steps: (1) initialize the ϵ -Machine; (2) homogenize the ϵ -Machine as defined by Eq. 1; (3) determinize the ϵ -Machine; (4) remove transient states from the ϵ -Machine.

The details of the above steps are briefly outlined as follows, where length-restricted versions of $\vec{S}$ and $\vec{S}$ are denoted as $\vec{S}^L$ and $\vec{S}^L$, with L being the length, and a past observation is represented as s^L . In step 1, through frequency counting, we compute an estimated probability distribution $\hat{P}^{(0)}(\vec{S}^1 | \vec{s}^0)$, where $\vec{s}^0$ represents the null sequence (thus the seemingly conditional probabilities are in fact independent probabilities). The subsequent step (step 2) contains multiple iterations with the main component directly following from Eq. 1. As L is increased by 1 in each iteration, a new causal state is created if a conditional probability $\hat{P}^{(L)}(\vec{S}^1 | \vec{s}^L)$ differs significantly (above a significance level α upon a hypothesis test) from all the conditional probabilities given the existing causal states. Otherwise $\vec{s}^L$ is absorbed into the existing causal state giving rise to the most similar conditional probability. This main component, along with its associated calculations, is repeated until the maximum history size L_{max} is reached. In the next step (step 3), each causal state is iteratively split into multiple states for each non-deterministic outgoing transition until all outgoing transitions are deterministic. A transition is deterministic if a transition out of one alphabet leads to only one causal state. This is repeated for all causal states. The transition table is computed here. We now have a deterministic machine. The aim of step 4 is to delete transient states. For each causal state, we iteratively check

each outgoing transition for any paths back to the source causal state. If there is none, the causal state is a transient state and is deleted. This is again repeated for all causal states. These steps eventually produce an approximation to the ϵ -Machine. In our experiments, we have chosen $L_{max} = 30$ and $\alpha = 0.01$.

After the ϵ -Machine is constructed, the probability distribution $P(\vec{S}^1|\Phi)$, where Φ is the ϵ -Machine's set of causal states, is computed for use by the POMDP.

3.3 Battery Modeling

The capacity fading effect and the capacity recovery effect are two important aspects in battery operation [6]. The former is captured by Peukert's Law, which describes the battery capacity as a function of the discharge rate. These two effects are largely attributable to a common underlying mechanism: in an electrochemical cell, electroactive species becomes depleted near the electrodes as the battery discharges; however, during an idle period, the internal diffusion of the electroactive species becomes the dominant process in the absence of the discharge current [7]. The diffusion results in what is called the capacity recovery effect. It is a stochastic process, with the ensemble probability having an inverse relationship with the extent of the charge depletion [6,7].

The battery's operation is modeled as a discrete-time process. During each interval of operation, the charge loss from the discharging operation is $c_d = mQ_d$, where Q_d is a constant base amount, and $m \in [0, 1, \dots, M_d]$ depends on the discharge current.

As a stochastic process, the recovery effect can be modeled as a Markov chain, as graphically illustrated in [6], [7] and [8]. A fully charged cell is assumed to have a maximum capacity of N charge units, where a charge unit is dependent upon both the battery discharge current (in mA) and the time resolution. E represents the charge level when the battery reaches the discharged state. During idle intervals, the battery recovers a charge c_r with a probability $P_r(c)$, where c is its current charge level. Following [6], we have set $c_r = \frac{Q_d}{2}$. We model the probability as

$$P_r(c) = \begin{cases} 0 & c = N \text{ or } c = E \\ \frac{1}{N-c}, & \text{otherwise} \end{cases}. \quad (2)$$

To an external decision maker, the battery's charge status is only observable through indirect measurement such as the voltage. The true status cannot be deduced from such indirect measurement without some uncertainty. However, this uncertainty is bounded. Thus, we model the partial observability of the battery as degradation by bounded noise. That is, the mapping function from the voltmeter reading to the underlying battery state is degraded by uniform additive observation noise.

3.4 Actions

The actions of the controller agent affect the battery states. By controlling variable resistances and shunt transistors, the agent makes a decision to attempt to

charge all or part of the PV-generated electricity to the battery while diverting the remaining current to the non-critical local load. Any amount exceeding the maximum local consumption capability in the time interval, however, is automatically dumped. The set of possible actions corresponds to controlling local electricity usage at different levels. The first possible action approximately corresponds to an attempt to charge all of the PV output to the battery, with the consequence that the battery's charge level increases if $c < N$. The other possible actions result in charging the battery with less amounts or discharging the battery. Possible battery discharges within any given interval is as described in Section 3.3.

3.5 Utility Function

When the battery is in state s_b , the immediate utility function (*i.e.* the *immediate* reward) is defined as follows:

$$u(c) = \begin{cases} -U_u, & c \leq E \\ -U_o, & c > N \\ \frac{U_n}{N-c+1}, & \text{otherwise} \end{cases}, \quad (3)$$

where $c = f_b^{-1}(s_b)$ is the charge level, with $f_b^{-1}(\cdot)$ being a mapping function; $U_u \geq 0$, $U_n > 0$ and $U_o \geq 0$ are constants. N is as defined in Section 3.3. When the utility value is positive, it represents a reward; otherwise it is a penalty.

3.6 System Setup

Solving POMDP in general for the infinite horizon is computationally intractable [9,10]. To maintain real-time computational tractability required for a system where the policy may need frequent updating, a maximum likelihood heuristic described in [11] is adopted here. The approximation that results from this has been shown to be sufficiently accurate when the uncertainty regarding the true state is due to bounded observation noise, rather than unknown state features [11]. In this method, policies for the underlying discrete-state MDP are computed first. Since the PV states are fully observable, the belief updating may be decomposed into two components, jointly described as (with the normalizing constant omitted):

$$b_t(s) \propto \delta(s_s - f_p(o_s))p(o_b | s_b, a) \sum_{s'_b \in S_b} \sum_{s'_s \in S_s} p(s_b | s'_b, s'_s, a)b_{t-1}(s'). \quad (4)$$

where $\delta(\cdot)$ is the Kronecker delta function, o_b is the battery charge estimated from the noisy voltage observation; $s \in S = S_b \times S_s$, where $s_b \in S_b$ is the battery's state, $s_s \in S_s$ is the PV's state, $f_p(o_s)$ is the result of discretizing the PV observation o_s (as described in Section 3.2).

The policy to be adopted is:

$$\tilde{\pi}(b, o_s) = \pi(\arg \max_s \{b_t(s)\}), \quad (5)$$

where $s_s = f_p(o_s)$, and $\pi(s)$ is computed based on the probabilities $\{P(\vec{S}^1 = s_s | \phi) : s_s \in S_s\}$, where $\vec{S}^1$ and Φ are as discussed in Section 2.2.

4 Experiments and Analysis

We have conducted tests using historical data and three methods. The data is the MPPT energy output (over 5-minute intervals) displayed in Figure 1. In our experiments, the initial beliefs are a uniform distribution over all of the states.

The first method employs random decision making. It assumes (1) no knowledge about the PV process and (2) no intelligence in the decision maker. As expected (see the top panel of Figure 2), this performed the worst; however, it is useful in establishing a baseline performance.

The second method for comparison also assumes no specific knowledge about the PV process. It uses a single-state transition matrix that is an identity matrix. This corresponds to zeroth-order hold prediction, where any observed state of the

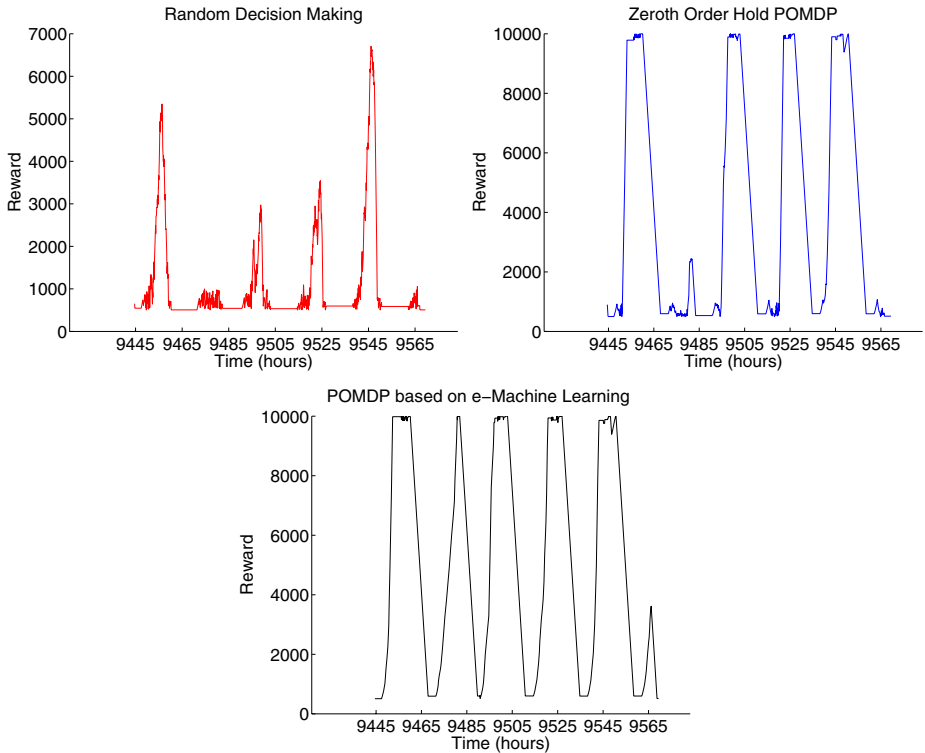


Fig. 2. Rewards (normalized battery capacities) resulting from random decisions (top left), the POMDP based on zeroth-order hold predictions (top right), and the POMDP based on the ϵ -Machine's learned knowledge (bottom)

Table 1. Comparison between a random decision making method, the POMDP with no knowledge of the PV process and the POMDP based on the ϵ -machine’s knowledge. The "Total Reward" displayed in the last row is the sum of the normalized battery capacities for each method.

	Random Decisions	Zeroth Order Hold POMDP	POMDP based on ϵ Learning
Process Knowledge	0	0	α
Intelligence in Decision Maker	0	1	1
Total Reward	1613243	5981586	7525413

PV output is supposed to be preserved as is into the next interval. The rationale for using this as a basis for comparison is that with dynamic signals resulting from an unknown underlying process, zeroth-order hold produces a prediction that is as accurate as it can be without analyzing the data history. In contrast to the first method, however, optimal decision making is employed with the use of a POMDP.

Next, an outcome that is largely stable, despite some of the low irradiance and high frequency changes, has been achieved by POMDP using the transition probabilities based on the ϵ -Machine’s learned knowledge.

Reward curves (normalized battery capacities) using the above approaches are displayed in Figure 2. In addition, a comparison between these approaches and their resultant accumulative rewards are summarized in Table 1. In that table, knowledge of the underlying process and the intelligence of the decision maker are rated on a scale between 0 and 1, and $\alpha < 1$.

5 Conclusion

Conventional peaking power plants add significantly to infrastructure cost, are not economic to build and maintain and produce greenhouse gases. In a scheme that uses PVs for both peaking power generation and local, non-critical consumption, we have investigated the possibility of integrating an ϵ -Machine with a POMDP. This approach has been motivated in part by the fact that efforts in extending traditional learning approaches to POMDPs have so far achieved only limited success. The PV module’s output has been assumed to follow a Markov process of a different order than the batter’s states, which are subject to the influence of the controller agent. Based on historical data, the ϵ -Machine learned all possible causal states up to a size of L_{max} . The partial observability of the battery status has been modeled as degradation by bounded noise. The other dimension, the output of a solar module, is fully observable. To maintain real-time computational tractability required for a system where the policy may need frequent updating, a maximum likelihood heuristic has been used. As the amount of solar irradiance usable by a PV module follows a stochastic process, we have compared the outcomes using different strategies in a stochastic environment,

namely, (1) employing a POMDP without any specific knowledge of the underlying processes, and (2) using a POMDP based upon learning the patterns of the underlying process. While in both cases the use of POMDP led to dramatic increases in the accumulative rewards, the addition of the process knowledge as discovered by the ϵ -Machine further gave rise to a substantial improvement. Whilst in this paper we have focused on local decision making, our target application and future work will involve multi-agent aggregation of distributed PV power generation.

References

1. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley-Interscience, Hoboken (1994)
2. Littman, M.L., Dean, T.L., Kaelbling, L.P.: On the complexity of solving Markov Decision Problems. In: Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI 1995), Montreal, Québec, Canada, pp. 394–402 (1995)
3. Shalizi, C.R., Shalizi, K.L., Crutchfield, J.P.: An algorithm for pattern discovery in time series. *Journal of Machine Learning Research* (to appear)
4. Shalizi, C.R.: Causal Architecture, Complexity and Self-Organization for Time Series and Cellular Automatas. PhD thesis, University of Wisconsin at Madison (2001)
5. Hussein, K.H., Muta, I., Hoshino, T., Osakada, M.: Maximum photovoltaic power tracking: an algorithm for rapidly changing atmospheric conditions. *Generation, Transmission and Distribution*, IEE Proceedings 142(1), 59–64 (1995)
6. Kim, J., Torregozza, J.P.M., Kong, I., Hwang, W.: Photovoltaic cell battery model for wireless sensor networks. *International Journal of Computer Science and Network Security* 6(9B) (2006)
7. Chiasserini, C., Rao, R.: A model for battery pulsed discharge with recovery effect. In: *Wireless Communications and Networking Conference*, pp. 636–639 (1999)
8. Rao, V., Singhal, G., Kumar, A., Navet, N.: Battery model for embedded systems. In: *18th International Conference on VLSI Design*, pp. 57–63 (2005)
9. Murphy, K.: A survey of POMDP solution techniques. Technical report, U.C. Berkeley (2000)
10. Meuleau, N., Kim, K.E., Kaelbling, L.P., Cassandra, A.R.: Solving POMDPs by searching the space of finite policies. In: *Proceedings of the Fifteenth Conf. on Uncertainty in Artificial Intelligence*, pp. 417–426 (1999)
11. Roy, N.: Finding Approximate POMDP Solutions Through Belief Compression. PhD thesis, Carnegie Mellon University (2003)

Using Numerical Simplification to Control Bloat in Genetic Programming

David Kinzett, Mengjie Zhang, and Mark Johnston

School of Mathematics, Statistics and Computer Science
Victoria University of Wellington, PO Box 600, Wellington, New Zealand
{kinzetalan,mengjie.zhang,mark.johnston}@mcs.vuw.ac.nz

Abstract. In tree based genetic programming there is a tendency for the size of the programs to increase from generation to generation, a process known as *bloat*. It is standard practice to place some form of control on program size either by limiting the number of nodes or the depth of the tree, or by adding a component to the fitness function that rewards smaller programs (parsimony pressure). Others have proposed directly simplifying individual programs using algebraic methods. In this paper, we add node-based numerical simplification as a tree pruning criterion to control program size. We show that simplification results in reductions in expected program size, memory use and computation time. We further show that numerical simplification performs at least as well as algebraic simplification alone, and in some cases will outperform algebraic simplification.

1 Introduction

In tree based Genetic Programming (GP) there is a tendency for the size of programs to increase, a process known as *bloat* [1,2,3,4]. This has a number of undesirable effects including increased memory usage and increased computation time. One of the main causes is the recombination via crossover. Standard practice is to give all nodes some chance of being the crossover point. Hence, as the programs become deeper (in levels), the average depth of the crossover point also becomes deeper, crossover is less likely to breed a child with a significantly different fitness, and thus the overall efficiency of the GP search is reduced.

A number of strategies have been proposed to combat bloat. Firstly, a limit can be placed on program size, limiting either the number of nodes or the depth of the tree [1,5,6]. This approach prevents bloat to some extent, but has some limitations as it is very difficult to set a limit a priori. If the problem is not already well known, then a series of trials will be required to establish what a good limit would be. This can be a significant extra computational cost in finding a solution. Also, when performing a crossover, trimming the subtrees being exchanged to fit the limit discards genetic material that may be important [5,7]. Secondly, a component can be added to the fitness function, or the selection process, that rewards smaller programs, a practice known as *parsimony pressure* [8,9,10,11,12,13]. This approach can be very successful in many problems, as it

goes some way to avoiding the loss of good programs that can occur with hard limits in that, if a program is good enough, then it will survive regardless of size. There are, however, situations when it can fail badly with all programs reducing to trivial sizes [14]. In fact, there are a number of reports on this approach that the effectiveness performance deteriorates [8,9]. Thirdly, a program can be simplified algebraically during the run, e.g., [7] uses efficient hashing techniques. This paper extends the program simplification approach by considering a simple form of algebraic simplification together with numerical simplification.

1.1 Research Goals

The goal of this research is to investigate a new program simplification approach to bloat control in tree based GP. We will consider two methods, one using simple algebraic simplification and the other with numeric simplification. We will compare the behaviour and performance of GP systems with both simplification methods and with no simplification. The specific questions we examine are:

1. Will the two simplification methods produce smaller programs than the canonical GP system with no simplification?
2. Will the two simplification methods reduce memory usage and shorten computation run times over the GP system with no simplification?
3. Which of the two simplification methods will perform better?
4. Will the simplification process affect the system performance?

2 Method

The aim of simplification is to remove redundant nodes and subtrees from programs. The *algebraic* simplification approach makes only those changes that leave the program producing exactly the same result as before the simplification, e.g., [7] use efficient hashing techniques. In our implementation we do not use hashing, but simply combine constant terminals, simplify the **IfPos** operator (see section 3.2) if the first argument is always strictly positive (or never strictly positive) across the training set, and simplify subtraction and division nodes if both subtrees are identical, replacing them with constant terminals of value zero and one respectively.

The idea of *numerical* simplification is to consider the numerical contribution that a node or subtree makes to the output of its parent node, removing those nodes and subtrees whose impact on the result is too small to make much difference to the program result. For efficiency reasons, this implementation addresses only the local effect of simplification at each node in the program tree. There will be cases where it does effect the system performance of the whole program, but the aim is to keep this to a minimum. It may be easiest to think of numerical simplification as a kind of lossy compression, where we aim to get useful reductions in program size without obvious loss in quality.

As the fitness is evaluated across the training set, each node keeps track of minimum and maximum values. The simplification process is performed from the

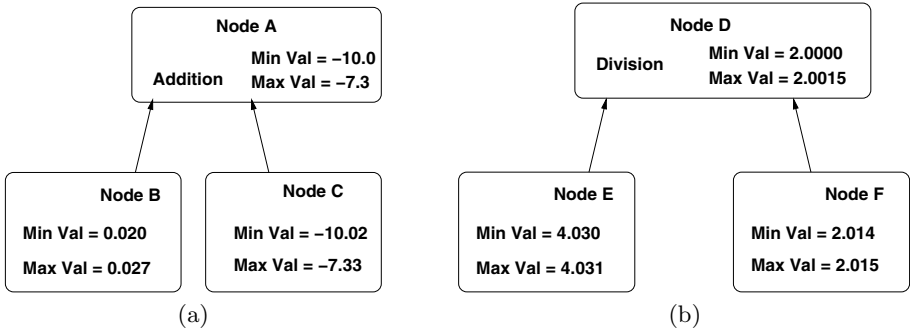


Fig. 1. Examples of numerical simplification

bottom up, so each operator is responsible for making those simplifications that are meaningful for it. A significance tolerance (threshold) is chosen (set at 0.001 in all our experiments). For addition and subtraction operators, a child node or subtree whose range of values is less than the threshold times the parent's minimum absolute value is discarded. Figure 1(a) gives an example of such kind. The range for Node B is $0.027 - 0.020 = 0.007$. The minimum absolute value for its parent Node A is 7.3. Since $0.007 < 0.001 \times 7.3$, the subtree headed by Node B will be discarded, and Node A will be replaced by Node C. Also, if the range of values a node takes is less than the threshold times its own minimum absolute value, the node is replaced by a constant terminal taking its average value. Figure 1(b) gives an example of this kind. The range for Node D is $2.0015 - 2.0000 = 0.0015$. The minimum absolute value for Node D is 2.0000. Since $0.0015 < 0.001 \times 2.0000$, the subtree headed by Node D will be discarded, and Node D will be replaced by a constant terminal with the value 2.00075. Note that the second type of simplification takes precedence over the first.

The numerical simplification method described here has the advantage that it is very simple and the process is embedded into the evaluation process, i.e., all nodes are evaluated only once.

3 Experimental Design

3.1 Experimental Datasets: Two Classification Problems

Coins. This dataset consists of a series of 64×64 pixel images of New Zealand five cent pieces against a random noisy background [7]. See Figure 2 for example head and tail images. There are 200 images of each of heads, tails and background only. Fourteen frequency features are extracted based on a discrete cosine transform of the image, as described in [15].

Wine. This dataset [16] gives the result of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantic ties of thirteen constituents found in each of the three



Fig. 2. Example images from the coin dataset (head and tail)

types of wines. These thirteen constituents are the features and the three classes are the cultivar from which the wine comes. This dataset was sourced from Weka project described in [17].

3.2 GP System Configuration

We considered three levels of simplification: no simplification, algebraic simplification only, and both algebraic and numerical simplification. The terminal set for each dataset consisted of the features used in that dataset and random “constant” numbers. The function set for both datasets consisted of the standard four arithmetic functions and the **IfPos** operator. The **IfPos** operator takes the value of the second child if the first child is greater than zero, and the third child otherwise. The fitness function used the error rate on the training set. Experiments are all conducted with the same set of population parameters. Some preliminary experiments established a set of parameters that gave reasonable classification performance without making too great a demand on memory resources for the no simplification case. The population size is 1000. Initial programs were five levels deep using the full method. Tournament selection was used with tournament size four. For the coin dataset we used 40 generations and for the wine dataset we used 200 generations. In addition, we applied two sets of parameters for the genetic operators: one 5% reproduction, 85% crossover, 10% mutation; and one 5% reproduction, 15% crossover, 80% mutation. We used ten-fold cross validation, and each combination of dataset, genetic operators and simplification method was run 200 times. For the various figures that follow, the runs were grouped in to sets of 20, with the results averaged over each of these 20 runs. Where simplification was used, it was performed after the first generation, and every fourth generation thereafter. This results in a periodic behaviour in program sizes as will be seen on the graphs that follow. Note that we intend *not* to set any maximum program size as the goal is to investigate whether simplification is sufficient to manage program bloat.

4 Experimental Results and Discussion

4.1 Analysis of Program Size: Number of Nodes

Figure 3 shows the number of nodes per program at each generation for the coin dataset. Each graph shows the maximum, mean and minimum program size for

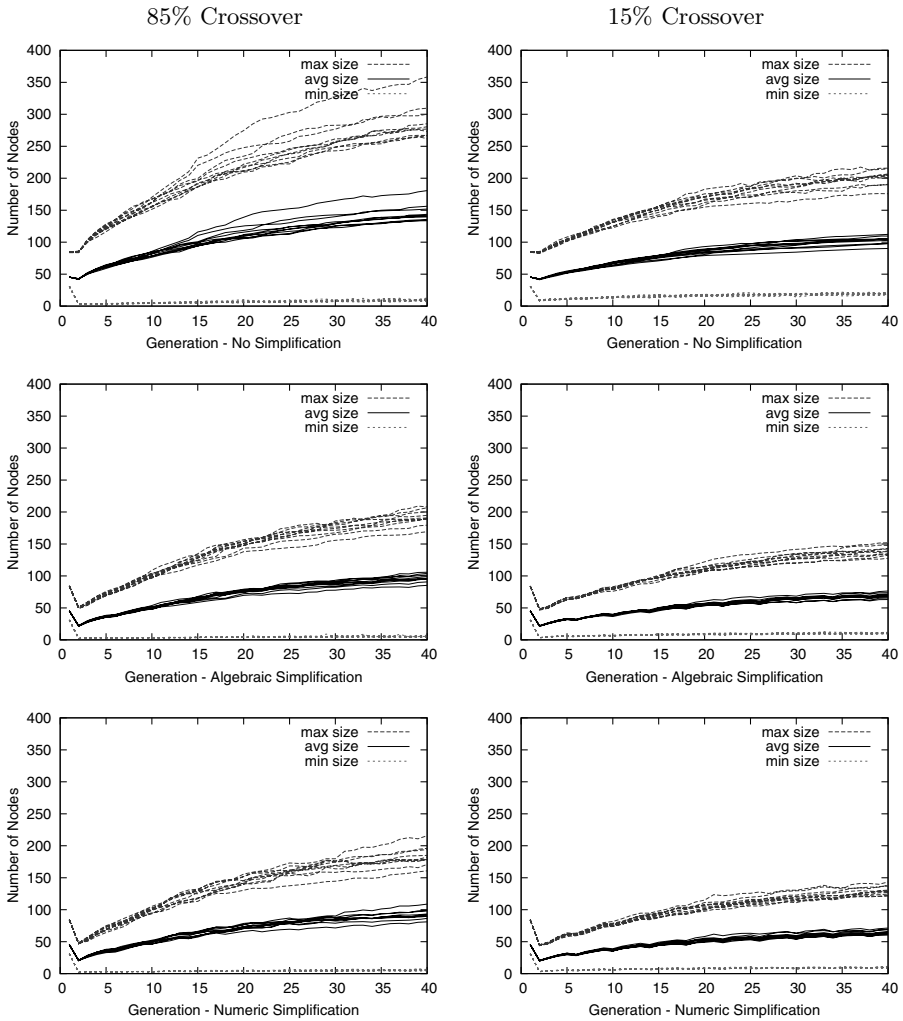


Fig. 3. Program size (number of nodes) for the coin dataset. In each graph, the maximum, mean and minimum program size is shown for 10 replications, each of which is the average of 20 runs.

10 replications, each of which is the average of 20 runs. The minimum program size is small with little variation. However, large program sizes are common (individual program size can reach 3500) which consequently gives large mean program sizes. It appears that program sizes are larger with 85% crossover, and continue to grow even with simplification, albeit more slowly than without simplification. Program sizes are smaller with the lower 15% crossover rate, and level off to a steady state with simplification, allowing the possibility of longer runs without the program sizes getting out of hand. This difference in performance is likely to be because the replacement subtrees used by the mutation

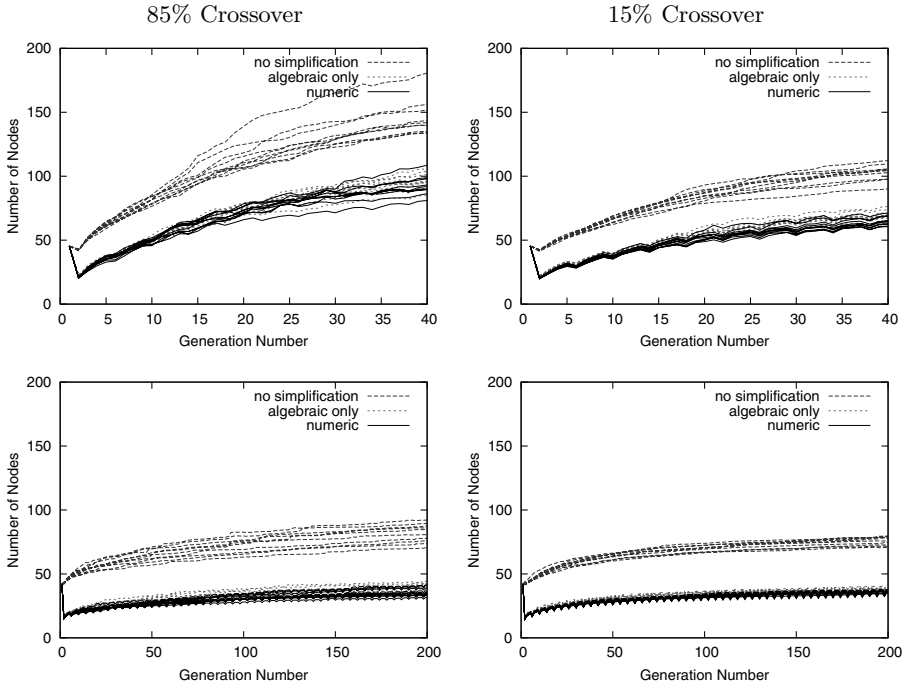


Fig. 4. Mean program size (number of nodes) for the coin dataset (top row) and wine dataset (bottom row). For each level of simplification, the mean program size is shown for 10 replications, each of which is the average of 20 runs. In both cases, the top collection of lines are all ‘no simplification’, and the bottom collection of lines are a mixture of ‘algebraic only’ and ‘numeric’ simplification.

operator are limited to a maximum depth of five levels and most are three or four levels. In contrast, the replacement subtree used by the crossover operator gets larger, on average, as programs become deeper. Therefore as the program depth increases, a mutation is likely to produce smaller offspring than the crossover operator. In all cases, the program sizes are significantly smaller with simplification than without. This is easier to see in Figure 4 (top row) which shows just the mean program sizes for all three simplification cases. Here we can clearly see the reduction in program sizes as a result of simplification. While both simplification methods show about a 40% reduction over no simplification, there is no noticeable differences between algebraic and numerical simplification. Finally, we observe the evident periodic nature of the curves, as a result of applying simplification every fourth generation, as described earlier.

The full results for the wine dataset are not included here because of page number limitations. The general nature of them are, however, very similar to those for the coin dataset as shown in Figure 3. Figure 4 (bottom row) shows the mean program size for all three methods for the wine dataset. Again, there appears to be no noticeable advantage to numerical simplification over algebraic simplification.

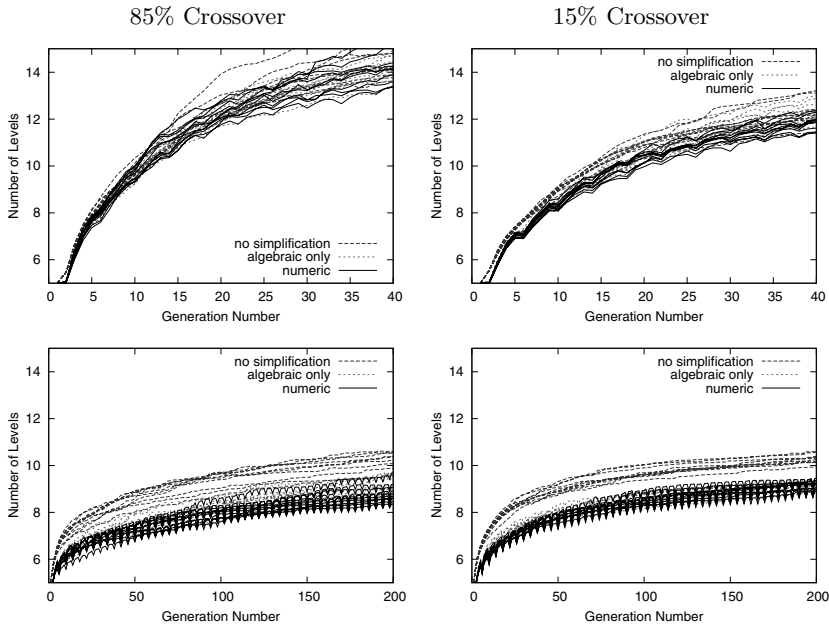


Fig. 5. Mean tree depth for the coin dataset (top row) and wine dataset (bottom row). Again these are 10 replications (curves), each of which is the average of 20 runs.

4.2 Analysis of Program Size: Tree Depth

Figure 5 shows the mean tree depth by generation for the two datasets. It appears that simplification produces shallower trees. With the coin dataset, it appears that there is no clear difference between the methods. The wine dataset shows a much clearer difference between no simplification and simplification, but no clear difference between the two simplification methods.

4.3 Analysis of Resource Usage

Figure 6 shows boxplots for memory use and CPU time. Some pattern is evident here, with simplification giving a noticeable reduction in CPU usage. There is also a reduction in memory usage, with simplification showing a reduction in memory usage of 30–40% in most cases, and a small advantage of numerical simplification over algebraic simplification.

4.4 Analysis of Classification Performance

Reducing the program size is of no benefit if the classification performance suffers badly. Figure 7 gives the mean test scores of the best fitness program at each generation for the coin dataset, again showing 10 replications, each of which is the average of 20 runs. There is little noticeable difference in performance between

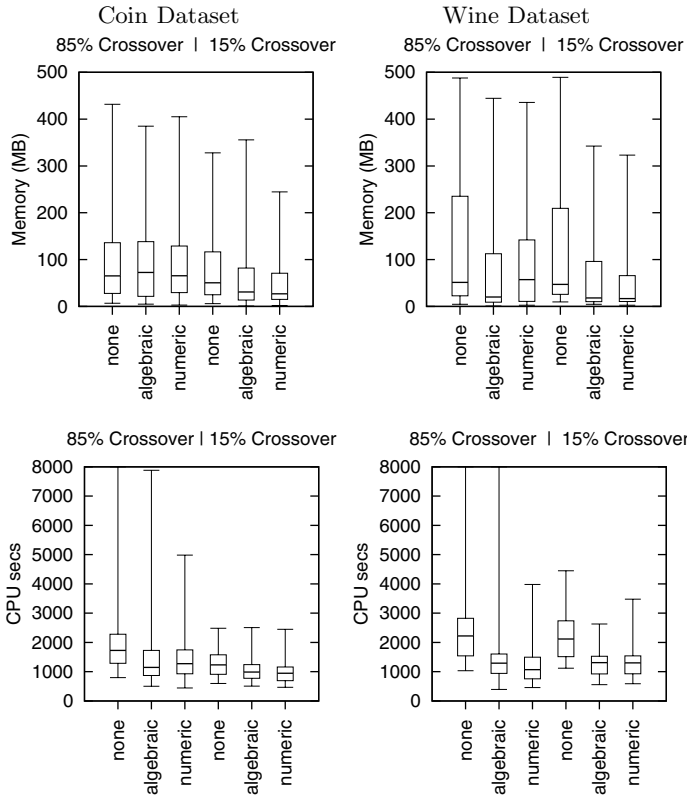


Fig. 6. Resource usage for the coin dataset (left) and wine dataset (right) for all 200 runs. The six boxplots on each graph are (left to right) 85% crossover with no simplification, algebraic simplification, and numerical simplification, then 15% crossover with no simplification, algebraic simplification and numerical simplification.

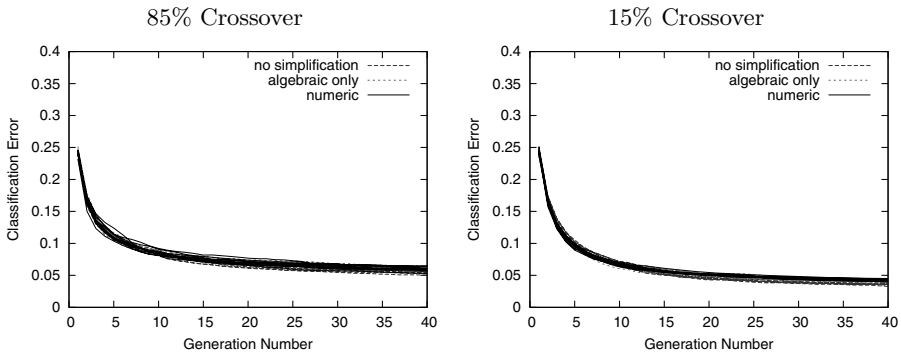


Fig. 7. Coin dataset classification performance

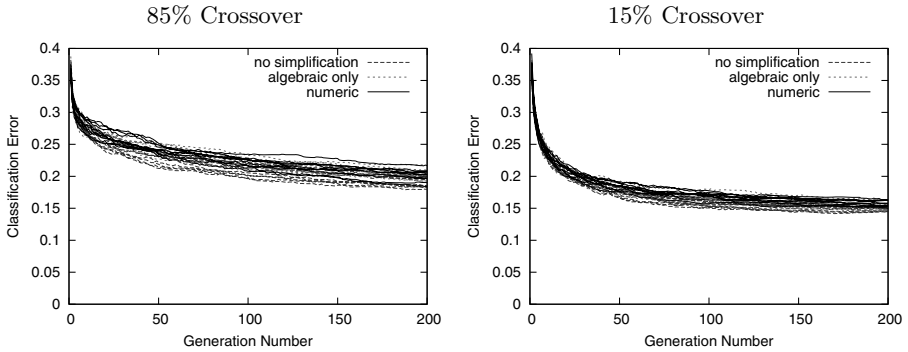


Fig. 8. Wine dataset classification performance

the three levels of simplification. Figure 8 gives the mean test scores for the wine dataset with similar results but with more variation in classification performance. The average performance in the graphs of Figure 8 is an classification error rate of between 15% and 20%, but some individual runs produced classifiers with zero error rate. The classification performance also appears to be sensitive to which examples are included the test set. Note that with both datasets, and both simplification methods, that the 15% crossover case produces both smaller programs and better test scores than the 85% case. This may be because of the smaller program size, or some other effect of the difference in genetic operators. More targeted experiments will be needed to establish this.

5 Conclusions and Future Work

The goal of this paper was to investigate a program simplification approach to bloat control in GP. The goal was successfully achieved by considering the effects of the tree nodes and subtrees and developing two simplification methods. The experiment results suggest that simplification certainly reduces the average number of nodes per program (by as much as 40%). This is partly due to reducing the depth of the tree, but much of the reduction is by thinning the tree. This correspondingly reduces memory use and run times, while having little effect on classification performance. Numerical simplification appears to be at least as effective as algebraic simplification alone (as expected), while showing some advantage in resource usage in many but not all cases. A key advantage of this approach over most of the existing approaches is that it does not require setting the maximum program depth or size for evolution and the simplification process can automatically control the possible code bloat and maintain the evolved programs within manageable sizes.

In future work, further targeted experiments will be needed to determine whether numerical simplification really improves upon algebraic simplification over a range of problem datasets and GP system configurations. The sensitivity of classification performance to the significance threshold will also be investigated. We will further investigate whether and how these simplification processes affect the building blocks for classification during evolution.

References

1. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
2. Soule, T., Foster, J.A., Dickinson, J.: Code growth in genetic programming. In: Koza, J.R., et al. (eds.) Genetic Programming 1996: Proceedings of the First Annual Conference, CA, USA, pp. 215–223. Stanford University, MIT Press (1996)
3. Soule, T., Heckendorn, R.B.: An analysis of the causes of code growth in genetic programming. Genetic Programming and Evolvable Machines 3(3), 283–309 (2002)
4. Blickle, T., Thiele, L.: Genetic programming and redundancy. In: Hopf, J. (ed.) Genetic Algorithms within the Framework of Evolutionary Computation, pp. 33–38 (1994)
5. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: Genetic Programming: An Introduction on the Automatic Evolution of computer programs and its Applications. Morgan Kaufmann Publishers, San Francisco
6. Zhang, M., Smart, W.: Using gaussian distribution to construct fitness functions in genetic programming for multiclass object classification. Pattern Recognition Letters 27(11), 1266–1274 (2006)
7. Wong, P., Zhang, M.: Algebraic simplification of GP programs during evolution. In: GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation, Seattle, Washington, USA, vol. 1, pp. 927–934. ACM Press, New York (2006)
8. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
9. Nordin, P., Banzhaf, W.: Complexity compression and evolution. In: Eshelman, L. (ed.) Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA 1995), Pittsburgh, PA, USA, pp. 310–317. Morgan Kaufmann, San Francisco (1995)
10. Parrott, D., Li, X., Ciesielski, V.: Multi-objective techniques in genetic programming for evolving classifiers. In: Corne, D., Michalewicz, Z., et al. (eds.) Proceedings of the 2005 IEEE Congress on Evolutionary Computation, Edinburgh, UK, vol. 2, pp. 1141–1148. IEEE Press, Los Alamitos (2005)
11. Zhang, B.T., Mühlenbein, H.: Balancing accuracy and parsimony in genetic programming. Evolutionary Computation 3(1), 17–38 (1995)
12. Zhang, M., Bhowan, U.: Program size and pixel statistics in genetic programming for object detection. In: Raidl, G.R., Cagnoni, S., Branke, J., Corne, D.W., Drechsler, R., Jin, Y., Johnson, C.G., Machado, P., Marchiori, E., Rothlauf, F., Smith, G.D., Squillero, G. (eds.) EvoWorkshops 2004. LNCS, vol. 3005, pp. 379–388. Springer, Heidelberg (2004)
13. Luke, S., Panait, L.: Lexicographic parsimony pressure. In: Langdon, W.B., et al. (eds.) GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 829–836. Morgan Kaufmann Publishers, New York (2002)
14. de Jong, E.D., Pollack, J.B.: Multi-objective methods for tree size control. Genetic Programming and Evolvable Machines 4(3), 211–233 (2003)
15. Marshall, D.: The discrete cosine transform (2001), <http://www.cs.cf.ac.uk/Dave/Multimedia/node231.html>
16. Forina, M., Leardi, R., Armanino, C., Lanteri, S.: Parvus: an Extendable Package of Programs for Data Exploration, Classification and Correlation. Elsevier, Amsterdam (1988)
17. Witten, I.H., Frank, E.: Data Mining: Practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann, San Francisco (2005)

Horn Query Learning with Multiple Refinement^{*}

Josefina Sierra¹ and Josefina Santibáñez²

¹ Universidad Politécnica de Cataluña, Spain
jsierra@lsi.upc.edu

² Universidad de La Rioja, Spain
josefina.santibanez@unirioja.es

Abstract. In this paper we try to understand the heuristics that underlie the decisions made by the Horn query learning algorithm proposed in [1]. We take advantage of our explicit representation of such heuristics in order to present an alternative termination proof for the algorithm, as well as to justify its decisions by showing that they always guarantee that the negative examples in the sequence maintained by the algorithm violate different clauses in the target formula. Finally, we propose a new algorithm that allows multiple refinement when we can prove that such a refinement does not affect the independence of the negative examples in the sequence maintained by the algorithm.

Introduction

In [1] Angluin, Frazier and Pitt proposed a query learning algorithm for the class of propositional formulas that can be represented by conjunctions of Horn clauses. The algorithm constructs a formula that is logically equivalent to the target formula to be learned using two types of queries: equivalence queries and membership queries. Query learning can be considered as a hybrid form of evolutionary learning, because the algorithm adapts its hypothesis in response to the evidence provided by the answers to membership queries and the counterexamples returned by equivalence queries.

In this paper we try to understand some decisions made by the Horn query learning algorithm proposed in [1]. We present a termination proof for such an algorithm that analyses the reasons why certain negative counterexamples returned by equivalence queries are discarded for refining some or all the examples of the sequence of negative examples maintained by the algorithm. In particular, we try to express the heuristics that underlie the decisions made by the algorithm in a declarative way [2,3], showing at the same time why such decisions guarantee at every moment that the elements of the sequence of negative examples maintained by the algorithm violate different clauses in the target formula.

We consider one of the main drawbacks of the algorithm proposed in [1,4]: the impossibility of using the negative counterexamples for multiple refinement. In order to guarantee the independence of the examples of the sequence S of

^{*} This work is partially funded by the DGICYT TIN2005-08832-C03-03 project.

negative examples, the algorithm uses the negative counterexample x returned by an equivalence query to refine only the first element of S that is refinable with respect to x . However it is not always necessary to refine a single element. We propose an algorithm that tackles this problem by allowing multiple refinement under certain conditions without giving up the independence of the negative examples of the sequence maintained by the algorithm, and we show that this algorithm is correct and terminates in polynomial time.

1 Basic Concepts

Definition 1. A propositional language L is specified by giving the set of its propositional symbols $\{p_1, \dots, p_n\}$. A *literal* is a propositional symbol p_i (a positive literal) or the negation of a propositional symbol $\neg p_i$ (a negative literal). A *clause* is a disjunction of literals. A *Horn clause* is a clause which contains at most one positive literal. A *Horn formula* is a conjunction of Horn clauses¹.

Horn clauses can be represented as implications. For example, the clause $\neg a \vee \neg b \vee \neg c \vee d$ is logically equivalent to the implication $a \wedge b \wedge c \rightarrow d$. Using the logical symbols T and F introduced above we can write Horn clauses such as $\neg a \vee \neg b \vee \neg c$ and a in implicative form as follows: $a \wedge b \wedge c \rightarrow F$ and $T \rightarrow a$.

Definition 2. Let C be a Horn clause. We define the antecedent of C , denoted $\text{antec}(C)$, as the set of propositional symbols that occur in the antecedent of the implicative form of C . Similarly the consequent of C , denoted $\text{conseq}(C)$, is the set of propositional symbols that occur in the consequent of C ².

Definition 3. An interpretation of a propositional language L is a mapping $x : L \rightarrow \{0, 1\}$ from the set of propositional symbols of L to the set of truth values $\{0, 1\}$. The evaluation of a propositional formula H with respect to an interpretation x , denoted $\text{eval}_x(H)$, is defined as follows (we assume that F and G are propositional formulas): (1) If H is propositional symbol p , then $\text{eval}_x(H) = x(p)$; (2) If H is of the form $\neg G$, then $\text{eval}_x(H) = 1 - \text{eval}_x(G)$; (3) If H is of the form $G \vee F$, then $\text{eval}_x(H) = \text{maximum}(\text{eval}_x(G), \text{eval}_x(F))$; and (4) If H is of the form $G \wedge F$, then $\text{eval}_x(H) = \text{minimum}(\text{eval}_x(G), \text{eval}_x(F))$.

Definition 4. Let H be a Horn formula of a propositional language L . An *example* is an interpretation x of the language L . An example x is *positive* (*negative*) for a propositional formula H if $\text{eval}_x(H) = 1$ ($\text{eval}_x(H) = 0$), i.e., the evaluation of H with respect to x is 1 (0). We use the notation $x \oplus$ ($x \ominus$) to denote the fact that x is a positive (negative) example for a given formula.

Definition 5. Let x be an example. We define $\text{true}(x)$ as the set consisting of the constant T and the propositional symbols which are assigned the truth value 1 by x . Similarly, $\text{false}(x)$ is defined as the set consisting of the constant F and the propositional symbols which are assigned the truth value 0 by x .

¹ We use the symbol T to denote the logical constant "true" (the propositional formula that is always true) and the symbol F to represent the logical constant "false".

² In particular, the antecedent of a Horn clause C without negative literals is $\{T\}$, and the consequent of a Horn clause D without positive literals is $\{F\}$.

Definition 6. Let s_1 and s_2 be two examples. We define the intersection $s_1 \cap s_2$ of both examples as the example z such that $\text{true}(z) = \text{true}(s_1) \cap \text{true}(s_2)$. Notice that this implies that $\text{false}(z) = \text{false}(s_1) \cup \text{false}(s_2)$.

The goal of the learning algorithm HORN proposed in [1] is to generate a Horn formula H that is logically equivalent to the unknown target Horn formula H^* to be learned. In order to construct a correct hypothesis H , the learning algorithm is allowed to gather information about the target formula H^* using two types of queries: membership queries and equivalence queries.

A *membership query* is a request for classification of a single example x with respect to the target concept H^* . The answer to a membership query is "yes" if x is a positive example for H^* , and "no" otherwise.

In an *equivalence query* the learning algorithm proposes a Horn formula H , which is the hypothesis it has constructed so far, to the oracle. If the current hypothesis H is logically equivalent to the target formula H^* , the answer to the equivalence query is "yes". In this case the algorithm halts and outputs H as the correct definition of the target concept. Otherwise, that is, if the current hypothesis is not logically equivalent to the target concept, the answer to the equivalence query is a *counterexample*, i.e., an interpretation $x: L \rightarrow \{0, 1\}$ which satisfies H^* but does not satisfy H (a *positive counterexample*), or an interpretation that satisfies H but does not satisfy H^* (a *negative counterexample*).

The answers to membership and equivalence queries can be found in polynomial time for a particular example x or hypothesis H and a target formula H^* , because the problems of evaluating a given formula with respect to an interpretation, and of determining whether two Horn formulas are equivalent (producing a counterexample if they are not) are solvable in polynomial time.

2 The Algorithm HORN

The algorithm HORN is based on the following idea: Every negative counterexample x returned by an equivalence query must violate a clause C of the target Horn formula H^* . The problem is that it is not possible to determine the exact form of C from the negative counterexample x alone. The only thing the algorithm can deduce about the violated clause $C \in H^*$ from the negative example x is that $\text{antec}(C) \subseteq \text{true}(x)$ and $\text{conseq}(C) \in \text{false}(x)$.

The approach used by the algorithm HORN to incorporate the information provided by each negative example x to its current hypothesis is to include in it all the clauses in the set $\text{clauses}(x)$, because each of these clauses is a possible explanation of why x is a negative example.

$$\text{clauses}(x) = \left\{ \left(\bigwedge_{v \in \text{true}(x)} v \right) \rightarrow z : z \in \text{false}(x) \right\}$$

The problem is that some of these clauses may be incorrect (i.e., not implied by H^*), or they may be correct but too weak. Incorrect clauses will eventually be discovered and removed from the current hypothesis when a positive counterexample that violates them is returned by an equivalence query.

A correct but weak clause C' of the set $\text{clauses}(x)$ is a clause with the same consequent as a clause $C \in H^*$ violated by x , but with an antecedent which may be much larger than the antecedent of C . This means that there will be many negative examples that violate C but satisfy C' .

The mechanism used to reduce the antecedents of those clauses that are correct but too weak is to attempt to refine the negative examples in which they are based with the negative counterexamples returned by subsequent equivalence queries. If the intersection of a previously obtained negative example with a negative counterexample returned by an equivalence query is negative (which can be checked using a membership query) and contains fewer true variables than the previously obtained negative example, such an intersection can be used as a refinement of the previously obtained negative example.

The algorithm HORN (shown in figure 1) works as follows. It maintains a sequence S of negative examples which is used to generate the hypothesis H used in each equivalence query made by the algorithm. The current hypothesis contains the conjunction of all the Horn clauses of the sets $\text{clauses}(x)$ for each negative example $x \in S$ that are not falsified by a previously seen positive example (lines 15 and 16 of the algorithm in figure 1).

Each negative counterexample returned by an equivalence query is either used to refine an element of S (line 12 of the algorithm), or is added to the end of S (line 14). In order to learn all the clauses of H^* , the algorithm makes sure that the examples in S violate different clauses of H^* . This is enforced using the following strategy: *Whenever a negative counterexample returned by an equivalence query can be used to refine several examples in the sequence S , only the first of these is refined.*

1. Set S to the empty sequence. $/^* s_i$ denotes the i -th element of S $^*/$
2. Set H to the empty hypothesis.
3. UNTIL (equivalent(H) returns "yes") DO {
4. Let x be the counterexample returned by the equivalence query.
5. IF (x violates at least one clause of H) THEN { $//$ x is a positive example
6. Add x to the set P of positive examples seen so far.
7. Remove from H every clause that x violates.
8. }
9. ELSE { $//$ x is a negative example
10. Let $\text{Refinable} = \{s_i : s_i \in S \wedge \text{true}(s_i \cap x) \subset \text{true}(s_i) \wedge (s_i \cap x) \ominus\}$.
11. IF ($\text{Refinable} \neq \emptyset$) THEN
12. Refine s_i replacing it with $s_i \cap x$, where $i = \min\{j : s_j \in \text{Refinable}\}$.
13. ELSE
14. Add x as the last element of the sequence S .
15. Set H to $\bigwedge_{s \in S} \text{clauses}(s)$ except for those clauses that are incorrect
16. according to the set P of positive examples.
17. }
18. }
19. Return H .

Fig. 1. The algorithm HORN

When a positive counterexample x is returned by an equivalence query, the algorithm adds x to the set P of positive examples seen so far and removes any clause violated by x from the current hypothesis (lines 6 and 7 of the algorithm).

3 Termination

First notice that the termination of the algorithm HORN implies its correctness, because the algorithm HORN terminates only if the hypothesis H and the target Horn formula H^* are logically equivalent.

Lemma 1. A negative counterexample x returned by an equivalence query with sequence of negative examples S cannot violate any clause $C \in H^*$ of the target formula such that $\text{antec}(C) = \text{true}(s_i)$ for $s_i \in S$.

Proof: A clause $C \in H^*$ such that $\text{antec}(C) = \text{true}(s_i)$ will belong to the current hypothesis H used in the equivalence query. Given that x is a negative example returned by such an equivalence query, x must satisfy every clause in H . $\square$

Lemma 2. A negative counterexample x returned by an equivalence query with sequence of negative examples S such that $\text{true}(s_i \cap x) = \text{true}(s_i)$ cannot violate any clause $C \in H^*$ such that $\text{antec}(C) \subset \text{true}(s_i)$ for $s_i \in S$.

Proof: Suppose that x violates a clause $C \in H^*$ such that $\text{antec}(C) \subset \text{true}(s_i)$. The current hypothesis H must contain the clause $\text{true}(s_i) \rightarrow \text{conseq}(C)$, since this clause is a logical consequence of C . But then we get a contradiction, because x violates the clause $\text{true}(s_i) \rightarrow \text{conseq}(C) \in H$ and we are assuming that x is a negative counterexample. $\square$

Lemma 3. A negative counterexample x returned by an equivalence query with sequence of negative examples S such that $\text{true}(s_i \cap x) \subset \text{true}(s_i)$ and $(s_i \cap x) \oplus$ cannot violate any clause $C \in H^*$ such that $\text{antec}(C) \subset \text{true}(s_i)$ for $s_i \in S$.

Proof: Suppose that x violates a clause $C \in H^*$ such that $\text{antec}(C) \subset \text{true}(s_i)$. Then $\text{antec}(C) \subseteq \text{true}(s_i \cap x)$ and $\text{conseq}(C) \in \text{true}(s_i \cap x)$, because $s_i \cap x$ is a positive example. However this leads to a contradiction. On one hand we have that $\text{conseq}(C) \in \text{true}(x)$, since $\text{conseq}(C) \in \text{true}(s_i \cap x)$. But on the other hand $\text{conseq}(C) \notin \text{true}(x)$, because we are assuming that x violates C . $\square$

Corollary 1. A negative counterexample x , returned by an equivalence query with sequence of negative examples S , that cannot be used to refine any of the negative examples in S should be added to the end of S because

1. x does not violate any clause $C \in H^*$ violated by any previous negative example $s_i \in S$ or any possible refinement s'_i of $s_i \in S$; and
2. x violates a clause $C \in H^*$ which cannot be induced using any previous negative example $s_i \in S$ nor any possible refinement s'_i of $s_i \in S$.

Although corollary 1 guarantees that step 14 of the algorithm HORN preserves the property that the elements of the sequence of negative examples maintained

by the algorithm violate different clauses, it does not imply that such a property is preserved during the whole execution of the algorithm. It is necessary to show that step 12 of the algorithm preserves that property, i.e., that any possible refinement of a negative example $s_i \in S$ cannot violate any clause $C \in H^*$ which is violated by a negative example $s_j \in S$ with $j < i$.

Lemma 4. Let S be the sequence of negative examples constructed by the algorithm HORN. At every moment during the execution of the algorithm S satisfies the following property: For every pair of distinct elements $s_i, s_j \in S$ and for every clause $C \in H^*$ if s_i violates C then s_j does not violate C .

Proof: Given corollary 1 we just need to show that whenever the algorithm HORN refines an element $s_i \in S$ with a negative counterexample x returned by an equivalence query with sequence of negative examples S , the refinement $s_i \cap x$ of s_i cannot violate a clause $C \in H^*$ such that $\text{antec}(C) \subseteq \text{true}(s_j)$ for some negative example $s_j \in S$ with $j < i$.

We demonstrate this result by induction on the total number of refinements k that have been applied to the elements of the sequence of negative examples S . If $k = 0$ the result follows from property 1 of corollary 1.

Suppose that $k > 0$ and that $s_i \cap x$ violates a clause $C \in H^*$ such that $\text{antec}(C) \subseteq \text{true}(s_j)$ for some $j < i$.

By induction hypothesis the sequence of negative examples S maintained by the algorithm HORN satisfied the property we want to demonstrate before the refinement of s_i took place. We assume that this was the last refinement (number k) applied to S . Therefore, s_i does not violate any clause $C \in H^*$ such that $\text{antec}(C) \subseteq \text{true}(s_j)$ for any $s_j \in S$ with $j < i$.

Given that $j < i$ it must be the case that either $\text{true}(s_j) = \text{true}(s_j \cap x)$, or $\text{true}(s_j) \subset \text{true}(s_j \cap x)$ and $(s_j \cap x) \oplus$, because otherwise the algorithm HORN would have refined s_j rather than s_i . But then lemmas 1, 2 and 3 imply that x cannot violate any clause $C \in H^*$ such that $\text{antec}(C) \subseteq \text{true}(s_j)$.

But this leads to a contradiction. Because if $s_i \cap x$ violates a clause $C \in H^*$ such that $\text{antec}(C) \subseteq \text{true}(s_j)$, either x or s_i must violate C . □

Lemma 5. If the target Horn formula H^* has m clauses, then at every moment during the execution of the algorithm the sequence of negative examples S does not contain more than m elements.

Proof: Since the elements of S are negative examples, each of them must violate some clause $C \in H^*$. However by lemma 4 two distinct elements of S cannot violate the same clause $C \in H^*$. Therefore S has at most m elements. □

Theorem 1. The algorithm HORN identifies every Horn formula H^* with m clauses and n propositional symbols in time polynomial in m and n using at most $O(m^2 n^2)$ equivalence queries and $O(m^2 n)$ membership queries.

Proof: During each execution of the main loop of the algorithm HORN one of the following things can happen:

1. A new negative example is appended to the current sequence of negative examples S (line 14 of the algorithm). This can happen at most m times, because S cannot have more than m elements.
2. An element $s_i \in S$ is refined replacing it with a new element $s_i \cap x$ which must contain strictly fewer propositional symbols assigned to 1 than s_i (line 12). This can happen at most n times for each element $s_i \in S$, because each s_i is an assignment of n propositional symbols to $\{1, 0\}$.
3. A positive counterexample returned by an equivalence query causes at least one clause to be removed from the current hypothesis H (line 7 of the algorithm). This can happen at most $m(n+1)$ times between modifications of S , since H cannot contain more than $m(n+1)$ clauses. This is so because S cannot have more than m elements and $\text{clauses}(s)$ contains $n+1$ clauses.

The sequence of negative examples S cannot be modified more than $m(n+1)$ times, because line 14 can be executed at most m times, and line 12 can be executed at most nm times (i.e., n times for each of the m elements of S).

On the other hand line 7 of the algorithm cannot be executed more than $m^2(n+1)^2$ times. Because each modification of S generates a hypothesis H which contains at most $m(n+1)$ clauses, and we have explained above that S cannot be modified more than $m(n+1)$ times.

Therefore the main loop of the algorithm HORN is executed at most $m^2(n+1)^2 + m(n+1) + 1$ times. Given that for each execution of the main loop of the algorithm one equivalence query is made, and either incorrect clauses are removed from the current hypothesis H or a modification of the current sequence of negative examples S is generated, the total time used by the algorithm is polynomial in m and n . This is so because evaluating the clauses in the current hypothesis H with respect to an interpretation x and removing incorrect clauses from the current hypothesis takes time polynomial in n and m . Checking whether any element of S can be refined with a negative counterexample x using at most m membership queries takes polynomial time as well. And generating a new hypothesis from the modified sequence of negatives examples S can also be done in time polynomial in n and m .

The maximum number of equivalence queries is $m^2(n+1)^2 + m(n+1) + 1$. Because a single equivalence query is made in each execution of the main loop of the algorithm. The number of membership queries is at most $m^2(n+1)$, because at most m membership queries are made every time S gets modified and we have explained already that S cannot be modified more than $m(n+1)$ times. $\square$

4 Multiple Refinement

The strategy adopted by the algorithm HORN in order to guarantee that the elements in the sequence of negative examples maintained by the algorithm violate different clauses in the target formula is the following: Whenever the negative counterexample returned by an equivalence query can be used to refine several elements of the sequence of negative examples maintained by the algorithm, only the first one of these is refined.

In this section we investigate conditions under which a multiple refinement strategy can be applied without giving up the property of independence of the negative examples in the sequence S maintained by the algorithm.

Let *Refinable* be the set of all the elements in the sequence S of negative examples maintained by the algorithm that can be refined using a negative counterexample x returned by the current equivalence query. That is, $\text{Refinable} = \{s_i : s_i \in S \wedge \text{true}(s_i \sqcap x) \subset \text{true}(s_i) \wedge (s_i \sqcap x) \ominus\}$. The proof of Lemma 4 shows that the refinement $s_i \sqcap x$ of an element $s_i \in S$ cannot violate a clause $C \in H^*$ which is violated as well by a non-refinable negative example $s_j \in S$ with $j < i$, i.e., by an s_j such that $j < i$ and $s_j \notin \text{Refinable}$.

This result, together with corollary 1, implies that in order to refine with x an element $s_i \in S$ which is not the first one in S that can be refined using x , we have to make sure that its refinement $s_i \cap x$ does not violate a clause $C \in H^*$ which is violated by the refinement $s_j \cap x$ of any refinable negative example s_j which precedes s_i in S , i.e., that for every $s_j \in \text{Refinable}$ with $j < i$, $s_j \cap x$ does not violate a clause $C \in H^*$ violated by $s_i \cap x$ as well.

Let $\text{Pre}^i = \{s_k : s_k \in \text{Refinable} \wedge k < i\}$ be the set of the refinable negative examples which precede s_i in S . The algorithm HORN-M, proposed in this paper, refines a negative example $s_i \in S$ using x only if it can guarantee that the refinement of s_i with respect to x cannot violate any clause which is violated as well by the refinement with respect to x of any negative example $s_j \in \text{Pre}^i$.

Different conditions can be used in order to guarantee that two negative examples violate different clauses of the target formula. The particular condition required by the algorithm HORN-M (shown in figure 2) is that their intersection be a positive example³. Thus before attempting to refine a negative example s_i with a negative counterexample x the algorithm makes sure that the intersection $s_i \cap x \cap s_j$ of the refinements of s_i and s_j (i.e., of $s_i \cap x$ and $s_j \cap x$) is a positive example for every $s_j \in \text{Pre}^i$ (see line 13 of the algorithm).

The following example shows how the algorithm HORN-M can be used to learn the propositional formula $H^* = \{a \rightarrow d, b \rightarrow c\}$, and compares its execution with that of the algorithm HORN proposed in [1].

Let $s_1 = 1010$ be the first negative counterexample⁴ returned by an equivalence query (we ignore positive counterexamples at the moment). The current hypothesis H built by the algorithm would be $H = \{a \wedge c \rightarrow b, a \wedge c \rightarrow d, a \wedge c \rightarrow F\}$.

Imagine that a second negative counterexample $x = 0101$ is returned by an equivalence query. As $s_1 \cap x$ is a positive example, x would be appended to the sequence S of negative examples as s_2 . The modified sequence of negative examples $S = \{s_1, s_2\}$ would generate the following hypothesis $H = \{a \wedge c \rightarrow b, a \wedge c \rightarrow d, a \wedge c \rightarrow F, b \wedge d \rightarrow a, b \wedge d \rightarrow c, b \wedge d \rightarrow F\}$.

³ If two negative examples s_i and s_j violate the same clause $C \in H^*$, then their intersection $s_i \cap s_j$ should violate C as well.

⁴ We describe the examples by giving the sequence of truth values assigned by them to the propositional symbols of the language sorted in lexicographic order. The example s_1 represents thus the truth assignment $s_1(a) = s_1(c) = 1$ and $s_1(b) = s_1(d) = 0$.

1. Set S to the empty sequence. /* s_i denotes the i -th element of S */
2. Set H to the empty hypothesis.
3. UNTIL (equivalent(H) returns "yes") DO {
4. Let x be the counterexample returned by the equivalence query.
5. IF (x violates at least one clause of H) THEN { // x is a positive example
6. Add x to the set P of positive examples seen so far.
7. Remove from H every clause that x violates.
8. }
9. ELSE { // x is a negative example
10. Let $\text{Refinable} = \{s_i : s_i \in S \wedge \text{true}(s_i \cap x) \subset \text{true}(s_i) \wedge (s_i \cap x) \ominus\}$.
11. FOR ($s_i \in \text{Refinable}$) DO {
12. Let $\text{Pre}^i = \{s_k : s_k \in \text{Refinable} \wedge k < i\}$.
13. IF $\boxed{(\text{Pre}^i = \emptyset \vee \forall j (s_j \in \text{Pre}^i \rightarrow (s_j \cap x \cap s_i) \oplus))}$ THEN
14. Refine s_i replacing it with $s_i \cap x$.
15. }
16. IF ($\text{Refinable} = \emptyset$) THEN
17. Add x as the last element of the sequence S .
18. Set H to $\bigwedge_{s \in S} \text{clauses}(s)$ except for those clauses that are incorrect
19. according to the set P of positive examples.
20. }
21. }
22. Return H

Fig. 2. The algorithm HORN-M

If a new negative counterexample $x = 1100$ is returned by an equivalence query at this point, the algorithm HORN-M would check that s_1 and s_2 are refinable with respect to x , because $x \cap s_1 = 1000, 1000 \ominus$ and $\text{true}(1000) \subset \text{true}(s_1)$, and $x \cap s_2 = 0100, 0100 \ominus$ and $\text{true}(0100) \subset \text{true}(s_2)$.

The algorithm HORN would refine only s_1 , because it is the first negative example in S that is refinable with respect to x . The algorithm HORN-M would refine s_1 as well, replacing it with s'_1 . But it would check whether the intersection $s_2 \cap x \cap s_1 = 0000$ is a positive example, and if it is so, it would refine s_2 with x as well. That is, it would update the sequence of negative examples S as follows $S = \{s'_1, s'_2\} = \{1000, 0100\}$,

The current hypothesis generated by HORN-M would be thus $H = \{a \rightarrow b, a \rightarrow c, a \rightarrow d, a \rightarrow F, b \rightarrow a, b \rightarrow c, b \rightarrow d, b \rightarrow F\}$ which only needs to be purged by the positive counterexamples generated by succeeding equivalence queries.

Termination

We demonstrate that for every target Horn formula with m clauses and n propositions the algorithm HORN-M terminates in time polynomial in m and n .

Lemma 6. Let S be the sequence of negative examples constructed by the algorithm HORN-M. At every moment during the execution of the algorithm S satisfies the following property: For every pair of distinct elements $s_i, s_j \in S$ and for every clause $C \in H^*$ if s_i violates C then s_j does not violate C .

Proof: Given corollary 1 we just need to show that whenever the algorithm HORN-M refines an element $s_i \in S$ with a negative counterexample x returned by an equivalence query with sequence of negative examples S , the refinement $s_i \cap x$ of s_i cannot violate a clause $C \in H^*$ which is violated as well by some negative example $s_j \in S$ with $j < i$.

We demonstrate this result by induction on the total number of refinements k applied to the elements of the sequence of negative examples S . If $k = 0$ the result follows from property 1 of corollary 1.

Suppose that $k > 0$ and that $s_i \cap x$ violates a clause $C \in H^*$ which is violated by some negative example $s_j \in S$ with $j < i$.

By induction hypothesis the sequence of negative examples S maintained by the algorithm HORN-M satisfied the property we want to demonstrate before the refinement of s_i took place. We assume that this was the last refinement (number k) applied to S . Therefore, s_i does not violate any clause $C \in H^*$ which is violated by any negative example $s_j \in S$ with $j < i$.

We distinguish two cases depending on whether $s_j \in \text{Refinable}$ or not. If $s_j \notin \text{Refinable}$, then it must be the case that either $\text{true}(s_j) = \text{true}(s_j \cap x)$, or $\text{true}(s_j) \subset \text{true}(s_j \cap x)$ and $(s_j \cap x) \oplus$. But then lemmas 1, 2 and 3 imply that x cannot violate any clause $C \in H^*$ such that $\text{antec}(C) \subseteq \text{true}(s_j)$.

But this leads to a contradiction. Because if $s_i \cap x$ violates a clause $C \in H^*$ which is violated by s_j , either x or s_i must violate C .

However, if $s_j \in \text{Refinable}$ we should take into account that the algorithm HORN-M must have refined s_j with x before attempting to refine s_i with x . This means that s_j must have been replaced with $s_j \cap x$ in S . We reason by reduction to the absurd. Suppose that $s_j \cap x$ and $s_i \cap x$ violate the same clause $C \in H^*$, then its intersection $s_i \cap x \cap s_j$ should violate C as well. But this is not possible, because the algorithm checks that $s_i \cap x \cap s_j$ is a positive example for every $s_j \in \text{Refinable}$ with $j < i$ before attempting to refine s_i with x . $\square$

Theorem 2. The algorithm HORN-M identifies every target Horn formula with m clauses and n propositions in time polynomial in m and n .

Proof: The proof of this theorem is analogous to the proof of theorem 1.

Alternative condition. In order to guarantee that two negative examples violate different clauses of the target formula the algorithm HORN-M checks that their intersection be a positive example. A weaker condition which still guarantees that two negative examples s_i and s_j violate different clauses in the target formula H^* is the existence of a positive example z such that⁵ $\text{true}(s_i \cap s_j) \subseteq \text{true}(z) \subset \text{true}(s_j)$. Therefore, we could modify the algorithm HORN-M replacing the condition used in line 13 with the following one, and still be able to prove its termination.

⁵ Suppose that s_i and s_j violate a clause $C \in H^*$, then $\text{antec}(C) \subseteq \text{true}(s_i \cap s_j)$. Let z be the positive example such that $\text{true}(s_i \cap s_j) \subseteq \text{true}(z) \subset \text{true}(s_j)$. It must be the case that $\text{conseq}(C) \in \text{true}(z)$, because z is a positive example. But then we get a contradiction, since $\text{conseq}(C) \in \text{true}(s_j)$ and we assumed that s_j violated C .

$$\text{Pre}^i = \emptyset \vee \forall j (s_j \in \text{Pre}^i \rightarrow \exists z (z \oplus \wedge \text{true}(s_i \bigcap x \bigcap s_j) \subseteq \text{true}(z) \subset \text{true}(s_i \bigcap x)))$$

The trouble is that checking such a condition might take time exponential⁶ in the difference between the number of 1's in the examples $s_i \bigcap x$ and $s_i \bigcap x \bigcap s_j$.

5 Conclusions

We have analysed the heuristics that underlie the decisions made by the Horn query learning algorithm proposed in [1]. Taking advantage of our declarative representation of such heuristics we have been able to provide an alternative proof for the termination of the algorithm, and to justify its decisions by showing that they always guarantee that the negative examples in the sequence maintained by the algorithm violate different clauses in the target formula. We have proposed a new algorithm that allows multiple refinement under certain conditions which do not affect the independence of the negative examples in the sequence maintained by the algorithm, and we have shown that this algorithm is correct and terminates in time polynomial.

References

1. Angluin, D., Frazier, M., Pitt, L.: Learning conjunctions of Horn clauses. *Machine Learning* 9, 147–164 (1992)
2. McCarthy, J.: *Formalizing Common Sense*. Papers by John McCarthy. Ablex. Edited by Vladimir Lifschitz (1990)
3. Sierra, J.: Declarative formalization of reasoning strategies: A case study on heuristic nonlinear planning. *Annals of Math. and Artif. Intelligence* 39(1-2), 61–100 (2003)
4. Balcazar, J.: Query learning of horn formulas revisited. In: *Computability in Europe Conference*, Amsterdam (2005)

⁶ Since there are $2^k - 1$ examples z such that $\text{true}(s_i \bigcap x \bigcap s_j) \subseteq \text{true}(z) \subset \text{true}(s_i \bigcap x)$, where k is the difference between the number of 1's in $s_i \bigcap x$ and $s_i \bigcap x \bigcap s_j$.

Evolving Digital Circuits in an Industry Standard Hardware Description Language

Jamie Cullen

Artificial Intelligence Laboratory
University of New South Wales, Sydney NSW
`jsc@cse.unsw.edu.au`
`http://www.cse.unsw.edu.au/~jsc`

Abstract. Evolutionary Meta Compilation (EMC) is a recent technique that enables unmodified external applications to seamlessly perform target program compilation and fitness evaluation for an Evolutionary Computation system. Grammatical Evolution (GE) is a method for evolving computer programs in an arbitrary programming language using a grammar specified in Backus-Naur Form. This paper combines these techniques to demonstrate the evolution of both sequential and combinational digital circuits in an Industry Standard Hardware Description Language (Verilog) using an external hardware synthesis engine and simulator. Overall results show the successful evolution of core digital circuit components. An extension to GE is also presented to attempt to increase the probability of maintaining an evolved program's semantic integrity after crossover operations are performed. Early results show performance improvements in applying this technique to the majority of the presented test cases. It is suggested that this feature may also be considered for use in the evolution of software programs in C and other languages.

1 Introduction

The idea of automatically generating computer programs dates back at least to the mid-1980's[1,2] and has grown considerably in popularity in recent years, due in no small part to Koza's work on Genetic Programming (GP)[3]. More recently, a competing evolutionary model of automatic program generation, known as Grammatical Evolution (GE), has also arisen. One of the reasons that GE is interesting in comparison to GP is that it potentially allows evolution to be performed in any programming language conforming to a context-free grammar, and is not restricted to the tree-like structures of LISP S-expressions[4]. To date, GE has normally been applied to the evolution of subsets of conventional programming languages, such as C, and toy languages tailored to a specific problem, rather than hardware designs. A brief introduction to GE is contained later in this paper. Further details are covered in [4] and [5].

Evolvable Hardware (EHW) has been with us for a similar amount of time as evolvable software[6]. High-Level Description Languages (HDLs) offer the Hardware Engineer similar advantages that high level programming languages offer

the Software Engineer over programming in machine code or assembly language: abstraction, re-targetability, modularity, etc. However, HDLs of the type used in industry (such as Verilog and VHDL) are normally highly complex software systems which include not only a programming language compiler or interpreter but also a circuit simulator and circuit synthesis engine, frequently conforming to large and complex standards (e.g. *IEEE 1364-2001*). Such complex standards present a challenge to engineers who wish to leverage some of the advantages of evolving in an HDL, and using standard trusted tools, but do not wish to develop and debug their own complex HDL system in order to do so. Scaling up to larger designs often requires more complex fitness evaluation, which may be more naturally and reliably performed in a real circuit simulator, than in a home-brew internal simulation system.

Evolutionary Meta Compilation (EMC) separates the target program compilation and fitness evaluation from the primary evolutionary computation system. External tools such `gcc` (in the case of evolving C programs), are used to compile/synthesize a target program, instead of using an internal home-grown language interpreter[7]. In typical applications an interprocess pipe is opened between the parent evolutionary computation system and the external tool. A testbench program is handed to the external tool and also used in conjunction with the target program to exercise the design under test and assign a fitness score. This score is then passed back to the primary evolutionary system via the interprocess communication mechanism. Advantages of the approach include freeing the developer from needing to develop and debug a complex language interpreter / compiler / simulator / synthesis engine etc., and in leveraging the availability of external tools which may be cost or time prohibitive to reimplement. Further details and initial feasibility experiments for both software and hardware evolution are covered in [7].

This paper will demonstrate how a GE-based system in conjunction with EMC may be used to evolve fundamental digital circuit building blocks in Verilog HDL. The use of EMC removes at least one barrier to entry in considering HDL-level evolution of such circuits: That of needing to roll one's own compiler / simulator / synthesis engine in order to test HDL-level programs. The use of GE allows us to readily specify a subset of a well-known HDL (Verilog), and then pass programs generated by GE to a pre-existing synthesis/simulation engine via EMC for compilation and fitness evaluation.

One of the key challenges inherent in the GE approach is the lack of semantic information encapsulated by the language description. A language grammar specified in Backus-Naur form can encapsulate significant syntactic information about a language, but leave out important semantic information, particularly state-holding information, which may prove useful in the evolution of computer programs or hardware specifications. This lack of state holding information becomes problematic in the evolution of circuits in Verilog HDL, particularly with structural designs, as crossover operations between valid genotypes can often result in semantically invalid children.

With such difficulties in mind, an extension to GE is presented which allows the recording of important state information, during the generation of the phenotype. In the spirit of the GE approach, the presented mechanism is largely target language independent: In a procedural language like C, it potentially enables the natural specification of language features such as global and locally scoped variables, as well as setting and jumping to branch points freely within the program space. In a hardware design language context, it enables the ready specification of commonly used design features, such as allowing the outputs of a module to feedback into inputs of an earlier instantiated circuit, or the automatic generation of names for instantiated components and wires, which remain valid even after crossover is applied.

Results indicate that this language feature can in some cases provide performance improvements over conventional GE in the evolution of hardware circuits. Under certain conditions, the feature represents an enabling mechanism: The system running with a “conventional” GE configuration was unable to find an optimal solution to the problem without the feature being enabled.

2 Background

2.1 Grammatical Evolution

Grammatical Evolution (GE) primarily differs from GP in that it uses Backus-Naur Form (BNF), or a close variant thereof, to evolve programs in an arbitrary language. GE also differs from GP in other ways: (1) GE uses linear genomes instead of a tree-based structure, (2) GE maintains a separation between the linear genotype and the phenotype, and during evolution GE uses a grammar to map the genotype to the phenotype. This phenotype, rather than the source genotype, is used as the target program to be run and its performance evaluated relative to its peers. A brief example of GE is given in this section (For further information on GE, see [4] and [5]). An example grammar is shown in Figure 1. This grammar represents a small subset of the programming language Verilog and can express simple structural circuits composed of `xor`, `or` and `and` gate primitives. A grammar that parses to a valid subset of Verilog was chosen to match the later examples and described applications, but in general any BNF compliant grammar could potentially be used.

The genotype in Figure 1 is used to walk the grammar rules, starting at the first rule. A codon from the genotype is used each time the grammar reaches a decision point - that is, a branch in the rule. Starting at rule `program` we generate “`module chmain ( x , y , ...` ” as the start of our phenotype. As `statements` is the only choice in the first rule, we descend into rule `statements` and do not use a codon yet. Rule `statements` has two choices, separated by a pipe symbol in the grammar. To choose which path to take we use the current codon value (143) and mod it with the number of choices. Since $143 \% 2 = 1$, option 1 (`statement statements`), rather than option 0 (`statement`) is selected. We descend into this new rule and have another decision point (3 choices of gate type). Using the next codon we get $40 \% 3 = 1$ so we take branch 1 again, which

<pre> <program> ::= module chmain (x , y , z , s , c) ; input x , y , z ; output s , c ; <statements> endmodule <statements> ::= <statement> <statement> <statements> <statement> ::= <and> <xor> <or> <xor> ::= xor <iname> (<output> , <input> , <input>) ; <or> ::= or <iname> (<output> , <input> , <input>) ; <and> ::= and <iname> (<output> , <input> , <input>) ; <input> ::= x y z <wname> <output> ::= s c <wname> <iname> ::= 10 11 12 13 14 15 16 17 18 19 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 <wname> ::= w0 w1 w2 w3 w4 w5 w6 w7 </pre>	<pre> Genotype: 143 40 121 224 156 118 18 143 ... module chmain (x , y , z , s , c) ; input x , y , z ; output s , c ; xor i25 (s , w0 , y) ; or i28 (c , z , w2) ; xor i26 (w0 , z , x) ; xor i12 (w6 , y , x) ; and i2 (w2 , x , y) ; endmodule </pre>

Fig. 1. Grammar (left) and Genotype with Resulting Phenotype (Right) for a Full Adder in Verilog

is xor. Our overall phenotype now looks like “module chmain (x , y , z , s , c) ; input x , y , z ; output s , c ; xor ...”. We move to the right and descend into **iname** and continue parsing as before. In general, when we reach the end of a rule, we pop back up to the previously called rule, and continue parsing that rule from the point where we left off. A stack is used to maintain the call history. The parse ends when we reach the end of the first rule and the stack is empty, or if we reach the end of the maximum genotype or maximum phenotype length. The resulting final phenotype from the example parse for a correct Full Adder design using this grammar is shown on the lower right hand side of Figure 1.

2.2 Problems with Crossover in Evolving Programs

In GE a convention single point crossover operator usually does not result in such a clean, in-space substitution of a sub-tree. When the second half of a parent is copied over to a generated child, the point at which the child program is in the grammar tree is different to that of the originating parent. This normally results in a different walk of the grammar tree beyond the crossover point in the child genotypes. Such crossovers seem wasteful as any useful solutions contained in the second half of the parents is now lost in the resulting children. To allow the preservation of such information, the “LHS” crossover operator was developed by Harper and Blair[8]. LHS is a two point crossover which uses the grammar in conjunction with the genotype to find an equivalent sub-tree in both parents. Essentially crossover is only allowed to occur at an equivalent rule boundary in both parents. “LHS” is short for “Left Hand Side” indicating that the crossover point must occur at a valid reference to a rule, rather than at any random point within a rule. e.g. For a specific crossover in the presented grammar above, we might randomly select the **statement** rule. Both parents genotypes are searched using the grammar until a **statement** rule is encountered. This point in both parents is recorded as the first crossover point (and may well be a different number depending on the parse). The rule in both parents is descended into until the parse returns back to the original rule. This second point is recorded for both parents and marks the end of the sub-tree. Crossover occurs between

these two points, guaranteeing an in-place substitution of a sub-string of the codons. This sub-string (representing the complete sub-tree) is descended into and return from by the rule, and swapped with the equivalent sub-string in the other parent. In a certain sense, the LHS crossover operator might be viewed as performing an equivalent role in GE to the regular sub-tree crossover operator used in Genetic Programming.

LHS crossover is useful in GE in reducing the number of invalid child programs generated by crossover. However, issues still remain in the production of valid child programs using this method. LHS crossover will essentially transplant a block of code and drop it into an appropriate place (same rule insertion point) in another program. However, there is no guarantee that the chunk of code is a self contained module that can be readily transplanted. Simple examples of this issue include the case of transplanting a block of C code from one program to another in which the variables used in donor program have not been declared in the recipient program. Similar issues exist in Verilog. For example, imagine two programs both containing a module instantiation labeled `i1`. Now imagine an LHS crossover in which module `i1` is transplanted from program A into program B (presumably not replacing the previous instance of the same name, but still placed within the same scope). Such a transplant would result in a semantically invalid Verilog program, due to the declaration of two modules of the same name in the same scope. Issues of this nature form a significant problem for GE-like approaches to evolving programs, but to date have not been effectively addressed. Such issues potentially limit the scalability of the GE approach in more complex domains where such variable usage may be desirable.

One approach we might consider to remove problems like this is to keep track of semantic information during the parse, and have such transpositions of code still result in semantically valid code. One way to specify or keep track of semantic information could be to use some type of Attribute Grammar[9]. Unfortunately, Attribute Grammars of the type often encountered in the literature are highly theoretical in specification and/or seemingly excessively complicated for what should be a relatively simple task: keeping track of declared variables/labels. A simple method that extends the BNF grammar in a largely language independent form, but which still allows the tracking of simple state information, such as declared variables, would seem to be a desirable extension to GE. Such a feature might also allow us to help place the GE feature set on an even footing with GP, while still presenting the advantage of using one's programming language of choice.

3 Methodology

3.1 Introducing DEFINE/USE Pairs

The basic form of the mechanism consists of adding two special non-terminal symbols to BNF: `DEFINE(X)` and `USE(X)`. These non-terminals may be used anywhere on the right hand side of a BNF statement i.e. after the `::=` rule assignment statement. Both take a non-optional parameter `X`. The parameter


```

<program> ::= module chmain ( x , y , z , s , c ) ; input x , y , z ; output s ,
c ; <defwires> <statements> endmodule
<defwires> ::= <defwire> <defwires> | <defwire>
<defwire> ::= wire DEFINE(WIRE) ;
<statements> ::= <statement> | <statement> <statements>
<statement> ::= <and> | <xor> | <or>
<xor> ::= xor DEFINE(INSTANCE) ( <output> , <input> , <input> ) ;
<or> ::= or DEFINE(INSTANCE) ( <output> , <input> , <input> ) ;
<and> ::= and DEFINE(INSTANCE) ( <output> , <input> , <input> ) ;
<input> ::= x | y | z | USE(WIRE)
<output> ::= s | c | USE(WIRE)

```

Fig. 2. Naive Full Adder Grammar demonstrating DEFINE/USE pairs

is used to hold a language-specific arbitrary label that mnemonically refers to the feature being implemented. Among other uses, the author has experimented with applying DEFINE/USE pairings to implement locally and globally scoped variables in C, branching to labels in C, and instantiate modules and wires in Verilog HDL. Primitive type checking using the same basic mechanism has also been attempted. In this paper we will focus on the hardware-related examples of declaring wires and gates using DEFINE/USE.

A simple example grammar for wires and gates using DEFINE/USE is shown in Figure 2. The given example is deliberately kept simple to aid in explanation of the technique. During the creation of the phenotype from the input grammar and linear genotype, DEFINE(WIRE) or USE(WIRE) may be encountered in the grammar parse. When a DEFINE is encountered, an additional variable slot is created during the program parse, and automatically labeled. For example, the first time DEFINE(WIRE) is encountered in a parse would cause a variable called `w0` to be added to a table of variables for the scope WIRE. The wire such defined might be called `w0`, the second `w1`, and so forth. The total number of such variables is recorded for later use in phenotype generation. When USE(X) is encountered during a parse, a note is made that a variable of the referenced score (WIRE in the case of USE(WIRE)) and the codon at that point saved¹.

Once the grammar tree traversal has completed, the linear phenotype is walked sequentially starting from its first token. Whenever a DEFINE(X) is encountered a new variable name is generated e.g. `w5` for the sixth such wire variable created (indexed at zero for the first variable), and inserted in in place of the DEFINE(WIRE) token. Whenever a USE(WIRE) token is encountered, we take the previously saved codon that follows the USE(WIRE) token and modulo divide it by the total number of variables. The resulting number is used to select which variable to use. The corresponding variable name is then inserted into the phenotype and the USE(WIRE) token and associated codon discarded. One might choose to visualize this as the creation of a dynamically generated rule:

¹ It is a matter of good form to create a grammar that guarantees that a DEFINE(X) is always presented in the grammar in all possible parses of the grammar tree *before* a USE(X) is encountered, regardless of LHS crossover operations. This makes it impossible to use a variable before any are defined. Grammars not following this rule of thumb may cause undefined behavior. The reverse of this is not true: DEFINES can be performed with no USES. This can be useful in the instantiation of components in Verilog, where module names may not need to be referenced directly, but do need to be uniquely declared.

`WIRE ::= w0 | w1 | w2 | ...`

Such a rule replaces the need to create a fixed wire definition rule, such as in Figure 1. A reference (**USE**) of the scope **WIRE** causes the dynamic **WIRE** rule to be selected, and the saved codon is used to pick an appropriate wire based on the number of options currently available in the rule. After a crossover operation occurs (normally using the structure preserving LHS operator), the **USE** and associated saved codon are used to pick a variable from the dynamic rule used for this particular program. In this manner, variable references are guaranteed to be valid, even though the variable reference in the recipient program may shift to a different variable than in the original donor program. Note that such a mechanism need not be confined to defining module names or wire definitions. For example, in C programs, variable declarations, labels and branch points, etc. could also be modeled in an analogous way.

3.2 Test Configuration

The **emc** system (an implementation of EMC combined with GE and additional features) was used to perform the evolutionary computation[7]. Icarus Verilog was selected for Verilog synthesis and simulation due to its ready availability, integrated synthesis engine, and steps towards compliance with *IEEE 1364-2001*.

Four basic digital circuits were originally chosen for evolution: A Full Adder design, and a 2bit x 2bit Multiplier, representing combinational circuit building blocks. A gated SR-Latch design and a Gated D-type Latch design, representing simple sequential circuit building blocks, were also used. Note however that SR-Latch results are not presented, largely due to space considerations and similarity to the D-latch test case. In all tests, conformance to a set of test vectors (based on an appropriate truth table) was selected as a suitable fitness function.

In this current set of presented results, designs were all evolved from one and two input gate primitives in a structural Verilog paradigm.

4 Results

Results for the Gated D-Latch Circuit are shown in Figure 3. Tournament selection with 20 individuals and a survival rate of 60% per tournament was used. The population was 500 individuals per generation for 1000 generations total. The operator set was LHS crossover followed by an immediate mutate (mutation rate = 1/50 per codon). A grammar using the **DEFINE/USE** feature set was created, and the fitness function was set to a variation of the truth table for the module, with a maximum possible score of 16. 25 random seed values were used. Figure 3 shows that one of the 25 runs achieved the maximum possible fitness score of 16. An additional set of runs was also performed where the grammar did not use this feature. In this later case, none of the runs reached the maximal fitness score.

Results for the Full Adder and Multiplier circuits are shown in Figure 4. For the Full Adder, a survival rate of 20%, instead of tournaments, was selected. The

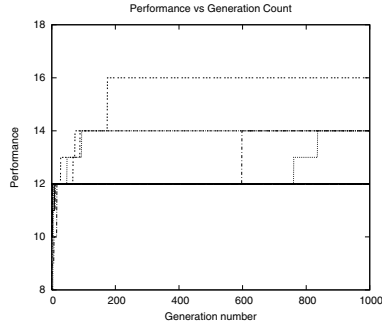


Fig. 3. Results for Gated D-Latch Circuit

remainder of the population was bred 50/50 split between mutation and LHS crossover. A mutation rate of 1/100 per codon was used, with a minimum of one mutation per genotype set to reduce the cloning of identical genotypes. As can be seen in Figure 4, performance is on average much better than the D-Latch design. A total of 24/25 runs managed to find an optimal solution of 16/16. When this same configuration was used with a grammar that did not utilize the DEFINE/USE feature, performance was lower, with only 17/25 runs finding the top score. A preliminary experiment to evolve a Full Adder circuit using behavioural (rather than structural) Verilog primitives was also performed under similar conditions, and successfully reached the optimal score. Details have been omitted due to space constraints. For the Multiplier, the configuration was similar to D-Latch, but with a smaller population size (200) and 50,000 generations. For this grammar, a key difference was that DEFINE was only used to avoid duplicate instance names, and implicit wire definitions were used instead of DEFINE/USE. A total of 4/25 runs manage to find an optimal solution of 64/64. The unenhanced grammar unexpectedly yielded higher statistical performance with 11/25 optimal solutions found. Due to the minimal use of DEFINE/USE, the careful structuring of the grammar to avoid unnecessary crossover in this test, and the impact on performance that a different grammar can have, it is difficult to draw any significant conclusions from this specific result. Additional runs, amongst other experiments, appear desirable to delve deeper into the statistical significance of these early results, but are currently unavailable at the time of publication.

It was observed in all test cases that the chosen grammar structure can make a significant difference in the ability of the system to escape local optima. For example, it was qualitatively observed that “unrolling” certain rules, such as those that provide gate and wire declarations, often provides a performance improvement. The rule of thumb derived from this observation was that it is useful for the grammar designer to make an educated guess as to how many wires and gates might be required, and to construct the rules accordingly to reflect these estimates. A hypothesis regarding this observation is that the presence of available “slots” in the design may in some cases enable the system to more readily

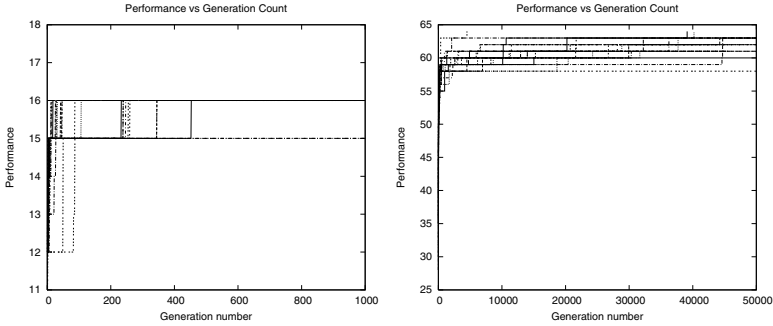


Fig. 4. Results for Full Adder (left) and 2bit x 2bit Multiplier Circuit (right)

escape local optima. It was also observed that grammars in which rules are constructed to minimize “pointless” crossover operations tend to offer performance benefits as well. For this reason implicit wiring was used in the multiplier design, and found to offer performance benefits in this case. With the multiplier design it was also found that additional gate slots (beyond the number needed to find a minimal solution) made a performance difference at the higher scoring range. To explain this phenomenon a hypothesis by way of an analogy to “puzzle pieces” is given: The extra gates and wires provide additional working space for multiplier to build an appropriate piece to complete the puzzle. When the design is near the optimal score, and additional space is not available, construction of the piece may result in the destruction of other parts of the design. This can result in reduced fitness of the piece-holding design, and an increased chance that it will be culled before having a chance to be connected to an appropriate piece. Additional slots appear to provide the space needed to non-destructively generate the last puzzle piece and complete the puzzle.

5 Conclusion

This paper combined the techniques of EMC and GE to investigate the evolution of both sequential and combinational HDL-based circuits. Results showed that functionally correct structural digital circuit primitives could be evolved for both classes of circuits, particularly when the presented DEFINE/USE extension was enabled. A preliminary run for behavioural language constructs was also successfully performed. Further research could examine the scalability of these approaches to more complex hardware designs, as well as more extensively explore different subsets of the Verilog language standard.

An extension to GE (DEFINE/USE pairs) was also presented. Examples were given for both wire and gate definitions in Verilog HDL. Initial experiments yielded successful solutions in all tests, as well as statistical performance improvements in the majority of the presented test cases. These early results appear promising, however more tests are required to investigate how generally

the feature can be applied, before broader conclusions may be drawn. As DEFINE/USE appears suitable to the evolution of software programs as well, future research might also quantitatively investigate applicability to other programming languages, such as C. Preliminary investigations have already been performed in this direction and suggest that DEFINE/USE may be used to model the basic features of local and global variables, labels and branch points, and perhaps also be used to provide a limited form of type checking. Expanding on such early explorations is the subject of future research.

Acknowledgments

Thanks to Alan Blair for his helpful suggestions and feedback.

References

1. Cramer, N.: A representation for the adaptive generation of simple sequential programs. In: Grefenstette, J. (ed.) *Proceedings of an International Conference on Genetic Algorithms and their Applications*, pp. 183–187 (1985)
2. Dickmans, D., Schmidhuber, J., Winklhofer, A.: *Der genetische algorithmus: Eine implementierung in prolog*. Technical report, Fortgeschrittenenpraktikum, Institut f. Informatik, Lehrstuhl Prof. Radig, Tech. Univ. Munich (1987)
3. Koza, J.: *Genetic Programming*. MIT Press, Cambridge (1992)
4. Ryan, C., Collins, J., O'Neill, M.: *Grammatical Evolution: Evolving programs for an arbitrary language*. In: Banzhaf, W., Poli, R., Schoenauer, M., Fogarty, T.C. (eds.) *EuroGP 1998*. LNCS, vol. 1391. Springer, Heidelberg (1998)
5. O'Neill, M., Ryan, C.: *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Springer, Heidelberg (2003)
6. Miller, J.F., Job, D., Vassilev, V.K.: Principles in the evolutionary design of digital circuits - part II. *Genetic Programming and Evolvable Machines* 1(3), 259–288 (2000)
7. Cullen, J.: *Evolutionary Meta Compilation: Evolving programs using real world engineering tools*. In: Hornby, G., Sekanina, L., Haddow, P. (eds.) *Evolvable Systems: From Biology to Hardware 2008* (2008)
8. Harper, R., Blair, A.: A structure preserving crossover in Grammatical Evolution. In: *2005 IEEE Congress on Evolutionary Computation*, pp. 2537–2544 (2005)
9. Knuth, D.: *Semantics of context-free languages*. In: *Theory of Computing Systems*, pp. 127–145. Springer, New York (1968)

Parameterised Indexed FOR-Loops in Genetic Programming and Regular Binary Pattern Strings

Gayan Wijesinghe and Vic Ciesielski

School of Computer Science and Information Technology,
RMIT University, GPO Box 2476V, Melbourne Victoria 3001, Australia
{[gayan.wijesinghe](mailto:gayan.wijesinghe@rmit.edu.au),[vic.ciesielski](mailto:vic.ciesielski@rmit.edu.au)}@rmit.edu.au

Abstract. We present two methods to represent and use parameterised indexed FOR-loops in genetic programming. They are tested on learning the repetitive unit of regular binary pattern strings to reproduce these patterns to user specified arbitrary lengths. Particularly, we investigate the effectiveness of low-level and high-level functions inside these loops for the accuracy and the semantic efficiency of solutions. We used 5 test cases at increasing difficulty levels and our results show the high-level approach producing solutions in at least 19% of the runs when the low-level approach struggled to produce any in most cases.

1 Introduction

Tree-based genetic programming (GP) has been widely and successfully used to evolve programs that have of a linear flow of execution, from the start to the end, and limited levels of flexibility. General programming constructs such as iteration and loops together with parameterisation have rarely been used within GP trees because they require conditional, repetitive execution and an emergence of logic that is not destroyed over the evolutionary process.

Loops are important for GP since some problems have repetitive characteristics and finding solutions to these problems without loops would be similar to writing the same instruction in the program body multiple times. One of the most common ways of solving such problems is by using an external loop in the environment that repeats the entire program, such as in the Santa Fe Ant problem [2].

This is quite different to human software engineering in procedural programming and it leads to inelegant and bloated programs. By allowing loops and parameters in GP, it is possible to represent a solution program that can scale with a problem but, achieving this is a very difficult task and reducing this difficulty is our primary focus in this work.

We are interested in evolving parameterised programs with indexed FOR-loops and we will investigate two formulations of these. To demonstrate our formulations, we wish to evolve programs that can learn the repeating unit of a given regular binary pattern string in a way that it can reproduce the pattern to arbitrary sizes. For example, consider the following regular binary pattern:

100110011001100110011001

The above is a string of length N , where $N = 24$ and it contains a well-defined repeating unit “1001” that regularly repeats an integer number of times. We wish to evolve a program with N as a parameter which will discover the repeating unit from a given set of examples and reproduce the pattern to arbitrary lengths N , within a specified range. This is a very hard problem since the programs must learn two closely related things, the repeating unit and how it is repeated, without destroying either finding over the evolutionary process.

Some of our preliminary work [9] has shown that the attempt to replicate procedural programming-like looping solutions within GP is very hard for the problem at hand as their operations are complicated and work at a low, character-level. We are interested in investigating how higher-level string operations affect the learning process in comparison to the programs that use lower level string operations. Specifically, we are interested in answering the following research questions:

1. How can we represent indexed FOR-loops in parameterised programs in genetic programming?
2. How can we use low-level operations within the indexed FOR-loops and how does it affect the learning?
3. How can we use high-level operations within the indexed FOR-loops and how does it affect the learning?
4. How do these methods compare in terms of learning accuracy and semantic efficiency in the regular binary pattern string learning problem?

Our motivation in this work is to get GP to produce scalable programs, narrowing the gap between the ability of human programmers and GP to advance the state of the art of machine learning.

2 Related Work

Some of the earliest work with loops was done by Kinnear, to evolve a sorting algorithm [3]. The author used a construct similar to a FOR-loop with an index and managed to produce solutions for sorting arrays of lengths up to 30. In this work, many training cases were used to improve solution scalability but no other mechanism was used in the fitness evaluation to promote this. Later Ciesielski and Li used a non-indexed FOR-loop construct in GP to evolve a sorting algorithm for a fixed-length array [2]. They did not focus on generality but rather the reliability by evaluating solutions on every permutation of the input data but at a fixed size.

Koza described a way to use loops in automatically defined functions (ADFs) [4]. However, there are very few reports of successful usage of this approach. Langdon used a type of ‘forwhile’ loops in ADFs to evolve the operations of a List data structure [5]. In their work, the fitness is based only on the tests that have been passed by a program, ensuring that a program need to completely and accurately learn every case to be a solution.

Pillay published a method that uses indexed FOR-loops to evolve a factorial calculating program and a program that draws a triangle on the screen [7]. However, this work does not state any mechanism that has been used to promote solution scalability or generalisation.

Li and Ciesielski used a domain dependant form of FOR-loops to solve a two class small binary image classification problem [6]. In their method, the loop traverses coordinates within images to perform feature calculations which are then used for classification purposes. This work has been extended by Wijesinghe and Ciesielski to classify relatively larger grey-level images that were transposed and rotated [8]. In both methods, the classifiers that used loops were more successful than the solutions without the loops. However, none of this work used index variables and relied on domain knowledge for correct functioning.

Several authors have published work on using loops for solving maze traversal problem such as the Santa Fe Ant problem, but without the use of an index [1,2].

3 A General Representation for Parameterised Indexed FOR-Loops in Genetic Programming

To answer our first research question, on representing FOR-loops in GP, similar to the ones found in procedural programming, we take a C language FOR-loop model and analyse its main components:

```
for (index initialisation; condition; index manipulation)
{
    loop body statement 1;
    loop body statement 2;
    ...
}
```

In the above, assume that the index variable is initialised to 0 and the continuation *condition* performs a boolean comparison to check if the *index* is less than a certain integer *TIMES*, i.e. $index < TIMES$. If we have this situation and the *index manipulation* component always increments the index variable by 1, the loop performs *TIMES* number of iterations. Therefore, changing the value of *TIMES* alone is enough to control the number of loop iterations. This model allows us to simplify the above, language specific yet fundamental, procedural programming looping model to a tree-based GP representation as follows:

(LOOP TIMES BODY)

Here, *LOOP* is the primary looping syntax function, similar to “for” in the C language syntax, and *TIMES* and *BODY* are its subtrees. As explained above, *TIMES* determines the number of times the *LOOP* will iterate its *BODY*. It is also a function of the output string length, *N*, which is the parameter that specifies the size of the problem. As a further simplification, just by finding the

appropriate divisors for N , GP can evolve the number of iterations required as it is often not necessary to iterate more times than N .

The *BODY* subtree contains one or more loop body functions. In this work, these body statements are string operations that reconstruct the output string and may operate relative to the value of the *index* variable of the loop. Multiple body statements can be included by having *PROG2* nodes as connectors that execute two *BODY* statements in order.

4 FOR-Loops with Low-Level Operations

When answering our second research question, how low level string operations can be used to learn the repeating unit of regular binary strings, we are interested in obtaining GP programs that are similar to the following C language program example:

```
for (index=0; index<(N/3); index++)
{
    tgt_string[index*3]=tgt_string[index*3]+1;
    tgt_string[index*3+2]=tgt_string[index*3+2]+1;
}
```

When the target string *tgt_string* has N elements initialised to '0', executing the above program would result in the repetition of the pattern "101". To allow such a representation in GP, we use strongly-typed tree-based GP and the following grammar:

```
LOOP      -> TIMES BODY
TIMES     -> TIMES / TIMES | <RAND_INT> | N
BODY      -> BODY BODY | ADD1
ADD1      -> TGT_POS
TGT_POS   -> TGT_POS+TGT_POS | TGT_POS*TGT_POS | <RAND_INT> | INDEX
```

As explained in the previous section, *LOOP* is the root node of the looping structure, similar to "for" in C-language syntax and the *BODY* is repeated *TIMES* number of times.

< *RAND_INT* > is a random number within the range 0 and the length of the longest pattern to be reproduced. N is the parameter which is the length of the target pattern that is to be reproduced. As the loop need not iterate more times than N , the only mathematical operations that we permit within *TIMES* are divisions.

The *BODY* subtree can contain one or more *ADD1* statements. We initialise all the characters of the string to "0" before the program is executed therefore requiring only "1" symbols to be added to reproduce the output. *ADD1* statements (explained below) add 1 to the character at the position in the string given by *TGT_POS*. *TGT_POS* is allowed to be a function of the index variable

(*INDEX*), random numbers ($< RAND\_INT >$) and mathematical addition and multiplication operators.

```
(LOOP (/ N 3) (PROG2 (ADD1 (* INDEX 3)) (ADD1 (+ (* INDEX 3) 2))))
```

The above is a GP solution that is equivalent to the C-language indexed FOR loop that was given earlier in this section.

The reason for our formulation of *ADD1* nodes is to ensure that each *ADD1* operation has a contribution to the final output. If we are to use an operator that destructively assigns a character in to the output string instead, the trees would bloat with many such operators that make no contribution to the final output, since the last such operation to execute at a particular element destroys the effect of the previous operation that has taken place. Our formulation could also make the GP crossover operations more objective since the final output is invariant of the *ADD1* node ordering within the trees.

The fitness evaluation requires the promotion of three aspects in the evolved programs. First, programs should produce output at arbitrary sizes. Then, the accuracy of the produced outputs needs to be evaluated to guide the evolution. We are also interested in obtaining clean programs with little or no invalid or inactive nodes due to semantic errors.

To promote the first aspect, we train the programs at multiple lengths, starting from the length of 1 repeating unit up to 10 repetitions of it at consecutive integer intervals. For example, if the repeating unit is of length 4, we evaluate the program at with the parameter $N == 4$, then 8, 12 and so on up to 40. In our preliminary investigations, we discovered that training on a higher number of repetitions does not necessarily improve the fitness of the final solutions.

To promote higher accuracy, we calculate the character distance difference between the reproduced and the original strings and refer to it as “accuracy error”. We calculate this for each different value of N , as a fraction of that N and then compute the average from the 10 different N values tested. The final values are within the range of 0.0 and 1.0.

To promote semantic efficiency and program correctness, we calculate the proportion of *ADD1* statements that operate outside the bounds of the output string to the total number of *ADD1* statements executed. This value is computed for each of the 10 values of N , averaged and is referred to as the “semantic inefficiency”. The final values of this measurement are within the range of 0.0 and 1.0.

The fitness of an individual is then the sum of the “accuracy error” and the “semantic inefficiency” but in the comparisons of our results, we are interested in the two components, individually.

5 FOR-Loops with High-Level Operations

To answer our third research question, we use a higher level representation for the string operators that reproduce the output string. Instead of using individual character-level operators such as *ADD1*, we use an approach that extracts an

entire substring from the original learning example and repeats it with respect to the length of the output string N .

The grammar for its representation is modified from the low-level method and is as follows:

```

LOOP      -> TIMES BODY
TIMES     -> TIMES / TIMES | <RAND_INT> | N
BODY      -> PATCH
PATCH    -> SRC_START SRC_LEN TGT_POS
SRC_START -> <RAND_INT>
SRC_LEN   -> <RAND_INT>
TGT_POS   -> TGT_POS+TGT_POS | TGT_POS*TGT_POS | <RAND_INT>
          | INDEX

```

The primary difference between the low-level formulation explained in the previous section and the high-level formulation given above is in the *BODY* subtree. Here, a *PATCH* statement is used now instead of having one or more *ADD1* statements. A *PATCH* is a function that extracts a substring that is *SRC_LEN* in length, starting from the *SRC_START* position in the example pattern and copies it to the target string, starting from the *TGT_POS* position.

PATCH is a higher level operation when compared to the *ADD1* operations described in the previous method since it is internally complex, performing a task that is equivalent to multiple *ADD1* statements and involves some looping behaviour within itself.

We use the same concept for the fitness function as we used in the low-level approach but due to the changed operations, it is necessary address some new issues. In the low-level formulation, there were only single character operations (*ADD1*) executed on the target string and therefore it was easy to prevent and keep account of the invalid operations that occurred outside the bounds of the target string. A modification to this method is necessary since we have entire substrings in this high-level formulation that might fall outside the bounds of the target string. Therefore, we do not allow the drawing or copying of an entire patch if any character of it would fall outside the bounds of the target string. However, only the individual characters that fall outside the bounds of the target string are counted as being invalid when calculating the semantic inefficiency component in the fitness function. In contrast, if a part of a *PATCH* source subscripts (*SRC_START* and *SRC_LEN*) falls outside the bounds of the source/example string, we wrap those subscripts around using the modulo function.

6 Test Cases and Experiment Setup

We use several test cases of increasing difficulty levels. Figure 1 shows these binary pattern units that we will attempt to learn and reproduce using our methods presented in this paper. These pattern units have been chosen such that they are not regular repetitions of smaller repeating units. Additionally, as

Problem	Pattern unit	Shortest example length	Longest example length
Length 2	10	2	20
Length 4	1001	4	40
Length 6	110001	6	60
Length 8	10100011	8	80
Length 10	1100010101	10	100

Fig. 1. The regular binary pattern problems

Table 1. Comparison of Final Results

Problem Length (<i>N</i>)	Accuracy Error		Semantic Inefficiency		Solution Count	
	L-level	H-level	L-level	H-level	L-level	H-level
2	0.002045	0.040246	0.000750	0.001041	79/80	52/80
4	0.197392	0.073716	0.255382	0.077952	2/80	28/80
6	0.256710	0.092069	0.037612	0.007116	0/80	15/80
8	0.226275	0.077242	0.002113	0.005954	0/80	23/80
10	0.262443	0.095426	0.008110	0.006100	0/80	16/80

the array elements are initialised to “0” and as our low-level method adds values (*ADD1*) in to those locations, the chosen pattern units of our problems contain 50% “0” symbols and 50% “1” symbols. For each problem, the programs will be trained at 10 different sizes between the given shortest and longest lengths.

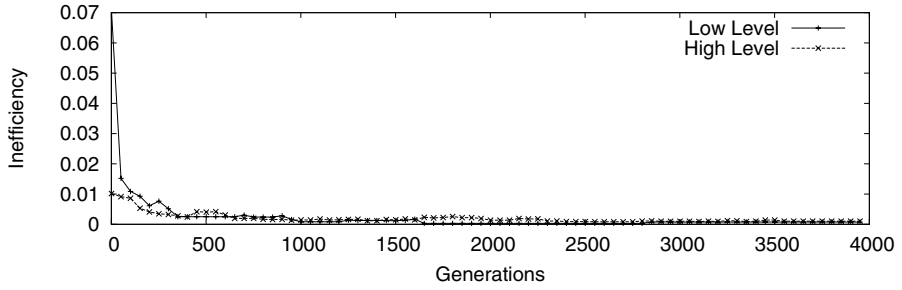
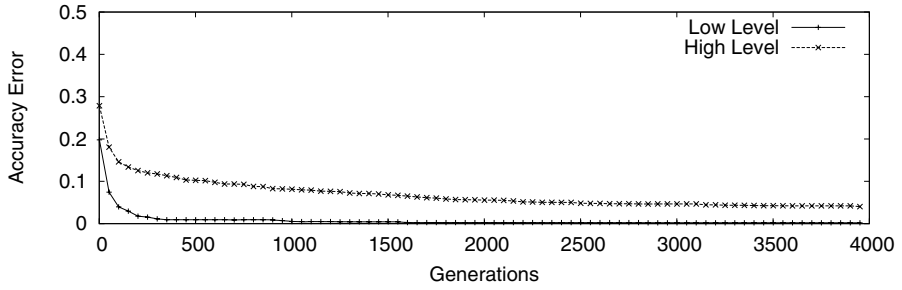
For clarity, we iterate that references to pattern unit lengths or the problem sizes (i.e. 2, 4, 6, 8 and 10) are different from the references to the various pattern lengths at which they are repeated in each problem.

For our GP setup, we use a strongly typed tree-based GP package. We use a population size of 100 stopping at a maximum of 4000 generations. Crossover, mutation and elitism rates are 70%, 28% and 2%, respectively. A maximum depth limit of 12 was forced on all trees to prevent any explosion in size.

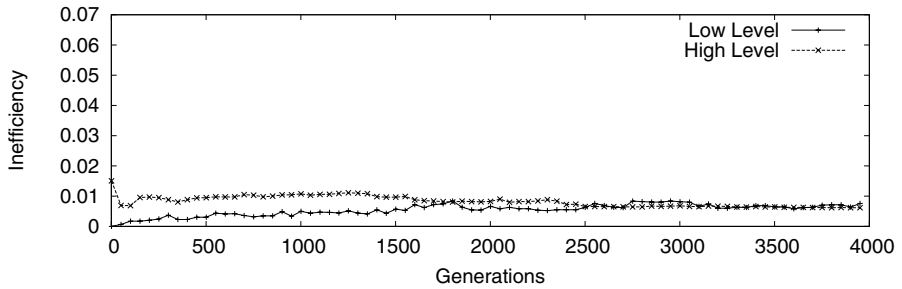
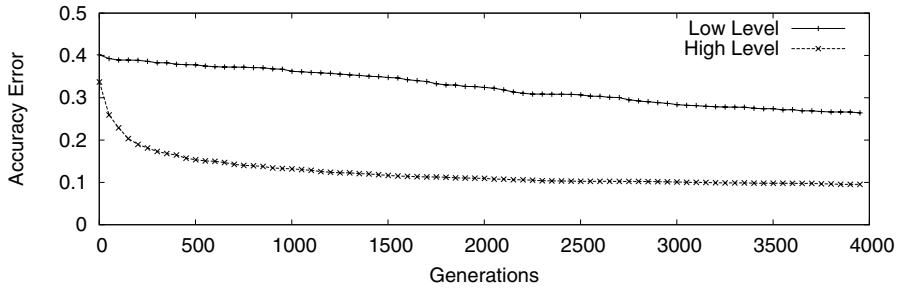
7 Results and Analysis

Table 1 shows the fitness values we obtained for all the problems and Figure 2 shows the change in the accuracy of the best individual and its semantic inefficiency during evolution in the smallest and the largest problems. The high-level formulation generally produced much better results except in the smallest problem size. When we analysed the runs we discovered that the low-level method’s solutions were found within the first 1000 generations. According to the grammars we used, the low-level approach only needed to search for a single item of information, the position of a single *ADD1* statement, whereas the high-level approach required the searching for 3 items of information (i.e. *SRC_START*, *SRC_LEN* and *TGT_POS*) that were necessary to its operation.

Although the high-level method performed worse in the smallest problem, it did not indicate an increase in accuracy error as the problem size increased.



(a) Length 2



(b) Length 10

Fig. 2. Fitness components of experiment lengths 2 and 10: Accuracy error and semantic inefficiency

We speculate that this is again due to the constant number of the items of information used in the high-level method's search. In comparison, the low-level method required its program bodies to grow in size depending on the number of 1 characters present in the pattern.

The following is a solution from the low-level method, found for the problem length 4:

```
(LOOP (/ (/ N 2) 2) ((ADD1 (+ INDEX (+ INDEX (+ INDEX INDEX))))
  (ADD1 (+ INDEX (+ INDEX (+ INDEX (+ INDEX 3)))))))
```

The following is a solution from the high-level method, found for the problem length 4:

```
(LOOP (/ N 4) (PATCH 4 4 (* INDEX 4)))
```

We also noticed that the semantic inefficiency (Figure 2) fluctuated over the generations for both approaches indicating that neither formulation has any advantages over the other in producing solutions that are more efficient.

The lack of solutions for problem sizes larger than 4 indicates that there is an inherent problem with the low-level method. A general observation that we have made here, after manually analysing many solutions closely is that, as the lengths of the problems become larger, the search arrives at a compromise or a sub-optimal solution that outputs a pattern that has similarities with the original pattern. Additionally, due to the formulation which requires a “cooperative” behaviour between *ADD1* statements, large parts of the *TGT_POS* subtrees needed to be similar under each *ADD1* statement, which greatly increased the difficulty of the search. Considering the destructive nature of the GP crossover operator, it is even more unlikely that such a cooperation will emerge easily. These were not issues that affected the high-level method, as can be seen by its performance.

8 Conclusions and Future Directions

We have presented and investigated two ways to use indexed FOR-loops together with parameters in GP. We applied these methods in learning the repeating unit of a regular binary pattern string in such a way that the pattern can be reproduced to user specified arbitrary sizes N . Our general approach involves using the problem scale variable, N , as a parameter inside the trees and training the programs for various values of N . The indexed FOR-loop constructs within the trees can then use this parameter along with string operations to reproduce the pattern to user specified arbitrary lengths.

We have investigated how low-level and high-level string operations within loops affect the quality of the produced solutions. The low-level approach consisted of an individual-character-level operator that adds a value to an evolved location that is repeated and changed by the loop. Using enough of these operators correctly within our indexed FOR-loops, a regular binary string can be

reproduced. However, this method was mainly successful in the smallest problem size due the complexity of its solutions.

The high-level approach differs from the above by extracting an entire substring from the example string and applying it to an evolved location that is repeated and changed by the loop. In this method, the size of the problem does not have a direct relation with the solution program size and therefore this method was successful in producing at least 1 solution per every 5 runs in all the tested problems.

As future work, we will investigate on how to improve and ensure the semantic validity of our solutions as we think it is a primary contributor to solution discovery and accuracy. We also plan on extending and applying the high-level method to 2-dimensional binary patterns.

References

1. Chen, G., Zhang, M.: Evolving while-loop structures in genetic programming for factorial and ant problems. In: Zhang, S., Jarvis, R. (eds.) *AI 2005. LNCS*, vol. 3809, pp. 1079–1085. Springer, Heidelberg (2005)
2. Ciesielski, V., Li, X.: Experiments with explicit for-loops in genetic programming. In: *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, Portland, Oregon, June 20–23, pp. 494–501. IEEE Press, Los Alamitos (2004)
3. Kinneer Jr., K.E.: Evolving a sort: Lessons in genetic programming. In: *Proceedings of the 1993 International Conference on Neural Networks*, San Francisco, USA, vol. 2, pp. 881–888. IEEE Press, Los Alamitos (1993)
4. Koza, J.R., Andre, D., Bennett III, F.H., Keane, M.: *Genetic Programming 3: Darwinian Invention and Problem Solving*. Morgan Kaufman, San Francisco (1999)
5. Langdon, W.B.: Data structures and genetic programming. In: Angeline, P.J., Kinneer Jr., K.E. (eds.) *Advances in Genetic Programming 2*, pp. 395–414. MIT Press, Cambridge (1996)
6. Li, X., Ciesielski, V.: Using loops in genetic programming for a two class binary image classification problem. In: Webb, G.I., Yu, X. (eds.) *AI 2004. LNCS*, vol. 3339, pp. 898–909. Springer, Heidelberg (2004)
7. Pillay, N.: Using genetic programming for the induction of novice procedural programming solution algorithms. In: *SAC 2002: Proceedings of the 2002 ACM symposium on Applied computing*, Madrid, Spain, pp. 578–583. ACM Press, New York (2002)
8. Wijesinghe, G., Ciesielski, V.: Using restricted loops in genetic programming for image classification. In: Srinivasan, D., Wang, L. (eds.) *2007 IEEE Congress on Evolutionary Computation*, Singapore, September 25–28, 2007, pp. 4569–4576. IEEE Computational Intelligence Society, IEEE Press (2007)
9. Wijesinghe, G., Ciesielski, V.: Experiments with Indexed FOR-Loops in Genetic Programming. In: *GECCO 2008: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, Atlanta, GA, USA, July 12–16, 2008, pp. 1347–1348. ACM, New York (2008)

Hierarchical Fuzzy Control for the Inverted Pendulum over the Set of Initial Conditions

Juliusz Zajackowski and Brijesh Verma

School of Computing Sciences, Faculty of Business and Informatics
Central Queensland University
Rockhampton, QLD 4702, Australia
jmzaj@yahoo.com, b.verma@cqu.edu.au

Abstract. We examine hierarchical fuzzy control of the inverted pendulum over the set of initial conditions. A new compositional method for the inverted pendulum system is introduced. Three layered hierarchical fuzzy logic topology is used to create a fuzzy rule base for the control system. Fuzzy rules are learnt by evolutionary algorithm designed for the compositional method.

1 Introduction

In this paper we investigate the compositional method applied to a typical dynamical system. The inverted pendulum system provides a good test platform used for the evaluation and comparison of various control theories. The control of the inverted pendulum has been undertaken using linear and nonlinear dynamics and include both classical and fuzzy logic control techniques, to mention a few, [1],[3],[5]– [10].

The major advantage of compositional method is that once the fuzzy rule base is determined it can control the dynamical system from a range of initial conditions without additional computations. In other words, the controller is fixed and is expected to successfully control the system from a large range of initial conditions.

A first step in the construction of a fuzzy logic system is to determine which variables are fundamentally important. In considering a single layered fuzzy system, with all variables as input into this layer, the total number of rules in a system is an exponential function of the number of system variables. This ‘curse of dimensionality’ can be handled in a variety of ways, one being the grouping of fuzzy rules into prioritised levels to design hierarchical structure.

In a hierarchical fuzzy logic structure (HFS), typically the most influential parameters are chosen as the system variables in the first level, the next most important parameters are chosen as the system variables in the second level, and so on. The number of rules in a complete rule set is reduced to a linear function of the number of variables. The decomposition into hierarchical fuzzy logic sub-systems reduces greatly the number of fuzzy rules to be defined and to be learnt. The task of finding the fuzzy rules in hierarchical fuzzy structure is given to evolutionary algorithm (EA).

The paper has been divided into six sections. The background of this research is given in Section 1. Section 2 describes the dynamical system under investigation.

Section 3 introduces the concept of hierarchical fuzzy systems and their application in control problem of the inverted pendulum system. In section 4 we give a detailed description of the evolutionary algorithm designed to find the fuzzy rule base. Details of computer simulations are given in Section 5. Final conclusions are drawn in Section 6.

2 Inverted Pendulum System

The nonlinear system to be controlled consists of the cart and a rigid pole hinged to the top of the cart. The cart is free to move left or right on a straight bounded track and the pole can swing in the vertical plane determined by the track. It is modelled by [11]:

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= u + m\ell(\sin(x_3)x_4^2 - \dot{x}_4 \cos(x_3))/(M + m) \\ \dot{x}_3 &= x_4 \\ \dot{x}_4 &= \frac{g \sin(x_3) + \cos(x_3)(u - m\ell x_4^2 \sin(x_3))/(M + m)}{\ell(4/3 - m \cos(x_3)^2/(M + m))}\end{aligned}$$

where x_1 is the position of the cart, x_2 is the velocity of the cart, x_3 is the angle of the pole, x_4 is the angular velocity of the pole, u is the control force on the cart, m is the mass of the pole, M is the mass of the cart, ℓ is the length of the pole, and g is gravitational acceleration. The control force is applied to the cart to prevent the pole from falling while keeping the cart within the specified bounds on the track. We take $m = 0.1\text{kg}$, $M = 1\text{kg}$, $\ell = 0.5\text{m}$, $g = 9.81\text{ms}^{-2}$, with state limits: $-1.0 \leq x_1 \leq 1.0$ and $-\pi/6 \leq x_3 \leq \pi/6$.

The goal is to determine fuzzy controllers necessary to stabilise the system about the unstable reference position $\tilde{x} = 0$ as quickly as possible, whilst maintaining the system within the target region (TR) defined by the following bounds: $|x_1| \leq 0.1$, $|x_2| \leq 0.1$, $|x_3| \leq \pi/24$, $|x_4| \leq 3.0$. The desired fuzzy controller is required to control the system such that the state variables converge to TR and are maintained within TR for a prescribed time limit T_f with $T_f = 20.0$.

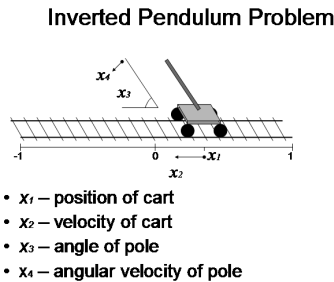


Fig. 1. Inverted Pendulum System

3 Hierarchical Fuzzy System

Our hierarchical fuzzy logic structure has two input variables in the first layer. Then there is one input variable in second and third layer of 3-layered HFS. This input configuration provides the minimal number of fuzzy rules in the knowledge base [12].

Investigation of different topologies for the inverted pendulum system showed that 3-layered topology with angular speed x_3 and angular position x_4 as input variables in the first layer, and then the cart's speed x_2 as input in second layer and position displacement x_1 as input variable in the third layer was a best choice for the control system (in terms of state variables convergence and control magnitude) [13].

The output variable u is calculated using the Mamdani product inference engine or minimum inference engine. Given a fuzzy rule base with M rules and n antecedent variables, a fuzzy controller as given in Equation 1 (with Mamdani inference engine) or Equation 2 (with minimum inference engine) uses a singleton fuzzifier, Mamdani product or minimum inference engine and centre average defuzzifier to determine output variables.

$$u = \frac{\sum_{\ell=1}^M \bar{u}^{\ell} (\prod_{i=1}^n \mu_{A_i^{\ell}}(x_i))}{\sum_{\ell=1}^M (\prod_{i=1}^n \mu_{A_i^{\ell}}(x_i))} \quad (1)$$

where $\bar{u}^{\ell}$ are centres of the output sets B^{ℓ} .

$$u = \frac{\sum_{\ell=1}^M \bar{u}^{\ell} (\min_{i=1}^n \mu_{A_i^{\ell}}(x_i))}{\sum_{\ell=1}^M (\min_{i=1}^n \mu_{A_i^{\ell}}(x_i))} \quad (2)$$

The architecture of 3-layered HFS is shown in Figure 2. For this system the first knowledge base KB_1 has the two inputs, x_i and x_j to produce as output a first approximation to the control u' . This u' together with x_k are used as input in the second

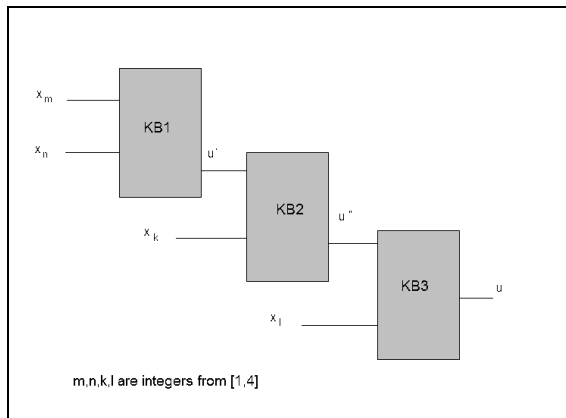


Fig. 2. Three Layer HFS

knowledge base KB_2 . Then the second layer produces another approximation of control u'' which with x_l is used as input to the third (and final) layer to produce the final control output u .

In the first layer there are $25 = (5)^2$ rules in the knowledge base. In general, we may write the ℓ^{th} fuzzy rule has the form:

$$\text{If } (x_i \text{ is } A_i^\ell) \text{ and } (x_j \text{ is } A_j^\ell) \text{ Then } (u' \text{ is } B^\ell).$$

where A_k^ℓ , $k = 1, 2, 3, 4$ are normalised fuzzy sets for input variables x_k , $k = 1, 2, 3, 4$, respectively, and where B^ℓ are normalised fuzzy sets for output variable u' .

For the second layer there are $35 = 7 \times 5$ rules in the knowledge base and we may write the ℓ^{th} fuzzy rule has the form:

$$\text{If } (u' \text{ is } C^\ell) \text{ and } (x_k \text{ is } A_k^\ell) \text{ Then } (u'' \text{ is } B^\ell).$$

where C^ℓ are normalised fuzzy sets for the input control variable u .

Similarly, there are 35 rules in the third layer and the ℓ^{th} fuzzy rule has the form:

$$\text{If } (u'' \text{ is } C^\ell) \text{ and } (x_l \text{ is } A_l^\ell) \text{ Then } (u \text{ is } B^\ell).$$

There are a total of 95 fuzzy rules in this hierarchical structure. The output for each layer is obtained using the either Mamdani inference engine or minimum inference engine as given in Equations 1 and 2, with the appropriate change of variable and associated membership functions for that variable.

Each domain region for x_i is divided into 5 overlapping intervals and each assigned membership sets: A_i^k , $k = 1, \dots, 5$, which we encode numerically as integers from 1 to 5 respectively. We found that the set of 5 membership functions provides relatively small knowledge base while maintaining a good controller performance. As the output variable range was found larger (by experiments), we divided the output u into 7 overlapping regions with 7 membership sets B^k , $k = 1, \dots, 7$, with integer encoding 1 to 7. All fuzzy membership functions are assumed to be Gaussian with their centres evenly spaced over the range of input and output variables.

4 Evolutionary Algorithm for Compositional Method

The compositional method uses the evolutionary algorithm to search for a fuzzy rule base capable of controlling the system over the whole set of initial conditions and fitness of each individual in the EA population reflects the controller performance for every initial condition in the set. In other words, every string in the population is assigned the fitness value which is a composite value representing string's performance for every single initial condition.

The evolutionary algorithm is a heuristic search technique that maintains a population of individuals $P(t)$ at iteration t to the next iteration $t + 1$ [14]. Each individual represents a potential solution to a given problem. Each individual is assigned a measure of fitness which defines how accurate it is as a potential solution to the problem. Depending on whether the problem is defined as a maximisation or minimisation problem, the best solution may be that string with the highest or lowest fitness value. The inverted pendulum problem is defined as minimisation problem.

Each individual string in the evolutionary population is to uniquely represent the hierarchical structure. This is achieved as follows. In the knowledge base of any layer, assuming a complete and consistent knowledge base, each fuzzy rule is uniquely defined by the consequent part.

This consequent part is identified by a particular output fuzzy set, for example, B^k . Such a fuzzy set can be identified by the integer $k \in [1, 7]$. The three fuzzy rule base structure can be represented as a linear individual string of $M = 25 + 35 + 35 = 95$ consequents, $p_k = [a_1, \dots, a_{95}]$, where a_j is an integer $\in [1, 7]$ for $j = 1, \dots, 95$.

The fitness f_k of a given string for a single initial condition is evaluated as follows: given an initial condition of the system we can decode each string p_k into the two or more components defining the fuzzy knowledge base for each layer, then use the Mamdani or minimum inference formula to evaluate u', u'' , and u to find the final control to be applied at each value of the state x . Given an initial state the system state equations are integrated by the Runge-Kutta algorithm (RK4) with step size 0.02 over a sufficiently long time interval $[0, T]$. The fitness f_k to be minimised, is then calculated based on some measures of the behaviour of the system over the time interval. These include, the accumulated sum of normalised absolute deviations of x_1 and x_3 from zero, the average deviation from vertical, the average deviation from the origin or $T - T_S$ where T_S , the survival time, is taken to mean the total time before the pole and cart break some bounds. A penalty of 1000 is added to the objective if the final state breaks the following bounds $|x_1| \leq 0.1$, $|x_2| \leq 0.1$, $|x_3| \leq \pi/24$, $|x_4| \leq 3.0$, i.e., leaves the designated TR (target region).

The fitness function f_k has the general form:

$$f_k = \omega_1 F_1 + \omega_2 F_2 + \omega_3 F_3 + \omega_4 F_4 + \omega_5 F_5 \quad \text{with}$$

$$F_1 = \frac{1}{N} \sum_1^N \frac{|x_1|}{x_{max}}, \quad F_2 = \frac{1}{N} \sum_1^N \frac{|x_2|}{\dot{x}_{max}}, \quad F_3 = \frac{1}{N} \sum_1^N \frac{|x_3|}{\theta_{max}},$$

$$F_4 = \frac{1}{N} \sum_1^N \frac{|x_4|}{\dot{\theta}_{max}}, \quad \text{and} \quad F_5 = \frac{1}{T}(T - T_S),$$

where $x_{max} = 1.0$, $\theta_{max} = \pi/6$, $\dot{x}_{max} = 1.0$, $\dot{\theta}_{max} = 3.0$, N is the number of iteration steps, survival time $T_S = 0.02 * N$, $T = 0.02 * N_{max}$ with the maximum number of iterations $N_{max} = 1000$, and ω_k are selected positive weights. The first and second terms determine the accumulated sum of normalised absolute deviations of x_1 and x_2 from zero, similarly for the third term and fourth terms in relation to x_3 and x_4 , and the last term when minimised, maximises the survival time.

The essence of the compositional method lies in the method of fitness function evaluation. The choice of evaluation method decides of effectiveness of the compositional method and therefore plays a crucial role. In our approach we decided to define a very simple evaluation method: the fitness function is evaluated for every initial condition and then averaged and assigned to a particular string in controller population.

Fitness function can be modified in order to reward strings which successfully control the system from a large number of initial conditions. One of the simplest methods is to establish threshold values for the objective function and penalize strings that exceed those threshold values (for each init. condition). We set the following threshold values:

$0.3 * avg$, $0.5 * avg$, and $0.8 * avg$ with corresponding penalties as: 500.0, 1000.0, and 2000.0, where avg is a variable representing average fitness of the previous generation:

```

Penalty schedule:
if ObjFun >= 0.3*avg and ObjFun < 0.5*avg
then ObjFun = ObjFun + 500.0;
if ObjFun >= 0.5*avg and ObjFun < 0.8*avg
then ObjFun = ObjFun + 1000.0
if ObjFun >= 0.8*avg then ObjFun = ObjFun + 2000.0

```

Please note, that increasing penalty values might ‘derail’ the evolutionary algorithm. Therefore penalties need to be fine-tuned to focus the EA on selecting strings that perform well for the large number of initial conditions. In some simulations the heavy penalties imposed on fitness function resulted in poor controller performance. To test the impact of above penalty scheme the same simulations were run without any penalties with mixed results that proved that usually not one factor decides on the EA performance but a combination of EA parameters.

The initial population, $P(0) = \{p_k : k = 1, \dots, M\}$, was determined by choosing the a_j as a random integer in $[1, 7]$ where M is the size of the evolutionary population. Full replacement policy was used and for selection process we used tournament selection with size $n_T = 4$.

The new population $P(t + 1)$ is obtained from the old one $P(t)$ by the use of genetic operators such as: selection, crossover, and mutation. Full replacement policy is implemented and requires that the population size remains constant from one generation to the next. A selection process is used to obtain parents for mating in the current generation. We chose tournament selection in which a number of possible parents are selected at random from the population. A tournament is then held to select the two most fit strings and they are used as parents in the next process of crossover to generate children to be passed into the next generation. We used tournament selection with size $n_T = 4$. An elitism strategy is typically used to pass the fittest individuals or copies of the fittest individual to the new population, so that the information encapsulated in the best individual is not lost and the fittest individuals are passed into the next generation.

In the crossover operation a number of ‘parent’ strings, typically two, are recombined to create ‘child’ strings. The children are then added to complete the new population. They also undergo mutation by a mutation operator which perturbs or mutates the string structures. With a given probability, usually small, the mutation operator mutates elements of the strings in the population. This ensures satisfactory diversity within the population which is required for the EA to find better approximate solutions to the problem. Mutation was undertaken with probability p_m whose value was determined by a mutation schedule that decreases typically from 0.8 to 0.001 over 1000 generations.

With an appropriate selection of EA parameters and operators, the algorithm is allowed to evolve. The most fit individual is then taken as the best possible solution learnt by this algorithm. Alternatively, the best top ten individuals are amalgamated (averaged) into the control system for the inverted pendulum. The described EA is used to learn fuzzy rules in the HFS that constitutes a control system for the inverted pendulum. A schematic algorithm is given below:

1. Population is randomly initialised: every component of individual string is given a randomly selected value from interval $[1, 7]$.

2. EA parameters are selected: type of inference engine, crossover, mutation schedule, selection method, elitism, fitness function (with a penalty schedule), and number of generations.
3. EA algorithm starts:
 - (a) Fitness of the first generation is evaluated.
 - (b) Next generation is created using EA operators: selection, crossover, mutation.
 - (c) Individual from the population is selected.
 - (d) Initial condition is selected from the predefined list.
 - (e) Dynamical system is simulated from a given initial condition.
 - (f) Final state of state variables and survival time are determined.
 - (g) Based on values from 3f temporary fitness function value is evaluated for an individual.
 - (h) Penalties are added to the fitness value (if penalty schedule is defined).
 - (i) Steps 3d–3h are repeated until all system simulations for every initial condition in the list are performed.
 - (j) Average of all temporary fitness values is calculated and assigned to the individual as its fitness.
 - (k) Steps 3c–3j are repeated until all individuals in the population have their fitness evaluated.
 - (l) Steps 3b–3k repeated until the final generation.
4. Final control system is determined by either selecting the top individual or by averaging top 10 individuals from the final population. Its performance is evaluated by running a simulation of the dynamical system for all initial conditions and counting initial conditions for which the final state variables are within the target region.

5 Simulation Results

Methodology described above was implemented in our experiments: 130 simulations were run for different variants of fitness function and different combinations of EA parameters. We illustrate our results on a few selected examples that represent typical results.

The population size was set at 500. Smaller population size is possible but below 300 it is difficult for the EA to maintain the required diversity in population. The EA was terminated at 1000 generations as it was found that the algorithm either finds solution in about 300 – 800 generations or fails. Changing the weights ω in fitness function had a significant impact on the EA performance. In fact, the fifth component of the fitness function ω_5 (survival time) implicitly contains all other components but by specifying them separately we influence the EA process, i.e., we introduce smaller or bigger bias towards one or another component. Full set of 255 initial conditions was defined for $|x_1| \leq 0.75$, $|x_2| \leq 1.0$, $|x_3| \leq \pi/12$, $|x_4| \leq 1.0$.

By introducing strong elitist strategy we achieved the convergence of the average value of objective function across population close to the minimum value of the objective function. This is desired as an indication of good EA performance resulting in majority of population being valid control systems. In several simulations the average population fitness was on par with the minimal fitness indicating that almost all individuals in the last population represented the control system of the same or very similar quality. In some simulations we amalgamated top ten control strings from the final population into one final controller. The amalgamation resulted in higher percentage of initial conditions from which the controller was able to control the system towards the

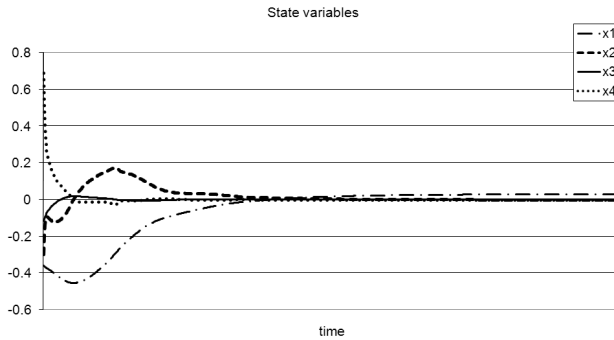


Fig. 3. Typical result: State variables x_k

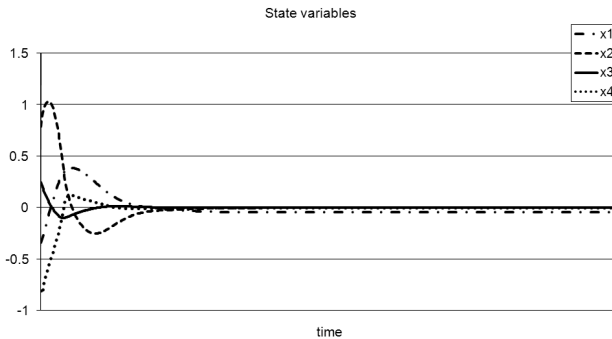


Fig. 4. Good convergence: State variables x_k

target area. The amalgamation of top control strings had adverse effect for the small number of initial conditions for which the controller went quickly out of range of numerical values ('blow-out' effect). With the controller defined as a top string from the final population control was always maintained, i.e., the trajectories converged if not in the target region than relatively close to it. However, as mentioned earlier, the number of initial conditions from which the controller successfully controlled the system to TR dropped. Typically, in most simulations the percentage of initial conditions for which controller successfully controlled the system to TR varied from about 40% to 60% with a bulk of simulations achieving just about 50% success rate. In many simulations the number of initial conditions from which the controller performance was satisfactory oscillated around 100 (out of 255). This trend might reflect the nature of the inverted pendulum dynamics. We observed that even though the controller often missed the target region the final state variables values were close to TR.

In simulation illustrated in Figure 3 minimum inference engine and penalty schedule were used. Results are shown for initial condition: $x = (-0.75, -1.0, -0.2618, -1.0)$ and: $\omega_1 = 3000$, $\omega_2 = 100$, $\omega_3 = 100$, $\omega_4 = 0$, $\omega_5 = 2000$. Another typical result is illustrated in Figure 3 for init. condition: $x = (-0.35, -0.5, -0.1309, 1.0)$ and:

$\omega_1 = 1000$, $\omega_2 = 0$, $\omega_3 = 1000$, $\omega_4 = 0$, $\omega_5 = 3000$. In simulation illustrated in Figure 4 we used product inference engine and no penalty schedule. Results shown are for initial condition: $x = (-0.75, -1.0, -0.2618, -1.0)$ and: $\omega_1 = 3000$, $\omega_2 = 100$, $\omega_3 = 100$, $\omega_4 = 0$, $\omega_5 = 2000$. In this particular simulation a very good state variables convergence was achieved but only for 99 out of 255 initial conditions. All state variables converged very quickly to zero, except x_1 with values remaining about 0.04 from the origin.

6 Conclusions

In this paper we have examined the hierarchical fuzzy control of the simple inverted pendulum and used evolutionary algorithms to learn a fuzzy controller for a single hierarchical topology over the whole set of initial conditions. We defined the set of 255 initial conditions in state space that is viable in terms of inverted pendulum dynamics. We excluded initial conditions that were in our opinion too extreme and would make control process virtually impossible. This fact reflects physical reality of inverted pendulum dynamics; if the system starts from completely different initial conditions it is unlikely to find relatively small fuzzy rule base capable of handling every possible dynamics of the system. For the set of this size and range, the task of controlling the inverted pendulum from every initial condition in the set proved unrealistic. We set our goal at achieving control to the target area for as many initial conditions as possible.

We designed the compositional method for the inverted pendulum system and proved that with the right combination of EA parameters the resulting fuzzy control system is capable to control the system from the wide range of initial conditions. Our results show that with improvements in the EA parameters (especially fitness function definition) better results are possible.

Compositional method provides a control system (not just a method for designing a controller) that once developed is capable of controlling the dynamical system from a wide range of initial conditions. In many practical applications this is a significant advantage as computational time for designing controller for every initial condition separately may render a method unfeasible.

References

1. Castillo, O., Cazarez, N., Rico, D.: Intelligent Control of Dynamic Systems Using Type-2 Fuzzy Logic And Stability Issues. *International Mathematical Forum* 1(28), 1371–1382 (2006)
2. Zajaczkowski, J., Stonier, R.J.: Co-evolutionary algorithm for hierarchical fuzzy control for the inverted pendulum. In: *Proceedings of WCCI 2006, IEEE International Conference on Fuzzy Systems (CD-ROM)*, Vancouver, Canada (2006)
3. Becerikli, Y., Koray Celik, B.: Mathematical and Computer Modelling. In: *Proceedings of the International Conference on Computational Methods in Sciences and Engineering 2004*, vol. 46(1-2), pp. 24–37 (2004)
4. Stonier, R.J., Zajaczkowski, J.: Hierarchical fuzzy controllers for the inverted pendulum. In: *Proceedings of CIRAS 2003*. Singapore, CD-ROM (2003)

5. Qiao, F., Zhu, Q.M., Winfield, A., Melhuish, C.: Fuzzy sliding mode control for discrete nonlinear systems. *Transactions of China Automation Society* 22(2), 313–315 (2003)
6. Suykens, J.A.K., Vandewalle, J., de Moor, B.: Optimal control by least squares support vector machines. *Neural Networks* 14, 23–35 (2001)
7. Yi, J., Yubazaki, N.: Stabilization fuzzy control of inverted pendulum systems. *Artificial Intelligence in Engineering* 14(2), 153–163 (2000)
8. Shuliang, L., Langari, R.: Hierarchical fuzzy logic control of a double inverted pendulum. In: *Fuzzy System 2000, FUZZ IEEE 2000, The Ninth IEEE International Conference*, vol. 2, pp. 1074–1077 (2000)
9. Huang, S.J., Huang, C.L.: Control of an Inverted Pendulum Using Grey Prediction Model. *IEEE Transactiona On Industry Applications* 36(2) (2000)
10. Stonier, R.J., Stacey, A.J., Messom, C.: Learning fuzzy controls for the inverted pendulum. In: *Proceedings of ISCA 7th International Conference on Intelligent Systems*, Melun, Paris, pp. 64–67 (1998)
11. Lin, C.T., Lee, C.S.G.: *Neural Fuzzy Systems*. Prentice-Hall, Englewood Cliffs (1996)
12. Wang, L.X.: *A Course in Fuzzy Systems and Control*. Prentice-Hall, Englewood Cliffs (1997)
13. Zajaczkowski, J., Stonier, R.J.: Analysis of hierarchical control for the inverted pendulum. In: *Proceedings of Complex 2004*. Cairns (CD-ROM), pp. 350–374 (2004)
14. Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolution Programs*, 2nd edn. Springer, Heidelberg (1994)

Genetic Programming for Feature Ranking in Classification Problems

Kourosh Neshatian, Mengjie Zhang, and Peter Andreae

School of Mathematics, Statistics and Computer Science
Victoria University of Wellington, P.O. Box 600, Wellington, New Zealand
{kourosh.neshatian,mengjie.zhang,peter.andreae}@mcs.vuw.ac.nz

Abstract. Feature ranking (FR) provides a measure of usefulness for the attributes of a classification task. Most existing FR methods focus on the relevance of a single feature to the class labels. Here, we use GP to see how a set of features can contribute towards discriminating different classes and then we score the participating features accordingly. The scoring mechanism is based on the frequency of appearance of each feature in a collection of GP programs and the fitness of those programs. Our results show that the proposed FR method can detect important features of a problem. A variety of different classifiers restricted to just a few of these high-ranked features work well. The ranking mechanism can also shrink the search space of size $O(2^n)$ of subsets of features to a search space of size $O(n)$ in which there are points that may improve the classification performance.

1 Introduction

Feature selection (FS) is a technique to optimise the performance of machine learning algorithms by finding an optimum subset of features which are sufficient to solve a problem. FS leads to dimensionality reduction by eliminating noisy and unnecessary features from the problem which in turn improves the performance and makes learning and execution processes faster. Models constructed using a smaller number of features are also easier to interpret.

Feature ranking (FR) is a common approach to feature selection in which features are ranked based on their usefulness and relevance to the problem [1]. An advantage of FR over other FS methods is that users have the freedom to choose an arbitrary number of features to obtain a good balance of accuracy, interpretation, dimensionality reduction and execution time based on their needs. Another benefit of FR is that it gives more information about the underlying nature of the problem by measuring the relative importance of each feature in learning. Most feature ranking methods fall into the filter approach category [2,3,4].

The capability of evolutionary algorithms (EAs) like genetic algorithms to explore complex search spaces has made them an efficient method for feature selection [5,6]. Genetic programming (GP) is a special class of EAs which is very flexible and expressive in dynamically building mathematical and logical models

based on an objective function [7]. Because GP programs/expressions are not bound to any predefined template and can be of any type (linear, non-linear, trigonometric, logical, etc), they can reveal different kinds of hidden relationships between the features of the input space and class labels.

GP has a built-in feature selection capability [8,9]. Similar to the process of optimising weights of features in neural networks, the importance of features in a GP search is revealed implicitly by incorporating useful features into construction of programs during the course of evolution. Therefore, using GP for feature selection can be categorised neither as a wrapper nor as a filter method [10]. In previous research, the use of features in the final program tree of GP classifiers has been an indication of the importance of those features [11]. A more sophisticated approach to using GP for feature selection has been taken by incorporating a multi-objective evolutionary optimisation method to encourage individuals with higher fitness and using lower number of features [12,13].

1.1 Goals

In this paper, we use the intrinsic capability of GP to implicitly select features, for feature ranking. In more concrete terms, the goals of this research are:

1. To use GP to rank the original features of a classification problem in a way that important/informative features get a higher rank while noisy and irrelevant features are ranked lower.
2. To shrink the search space by providing several subsets of features that can possibly improve the the performance of a classifier.

2 GP for Feature Ranking

We use GP to discover the hidden relationships between the features of a classification problem and the target class labels. Figure 1 depicts the overall architecture of the system. Given a dataset for a classification task, we conduct R parallel GP runs to construct a set of decision stumps for each class in the problem. Each run of the GP system will generate m “best” programs, one for each class label (system keeps separate populations corresponding to each class label). These decision stumps are weak classifiers which act like decision trees with a single node. The performance of these decision stumps are the fitness of the corresponding GP program and are calculated based on their performance in separating different class labels. The results of the GP runs (constructed decision stumps) along with their respective fitness are passed to the feature ranker component where the features are ranked based on the frequency of their appearance in the programs and the fitness of the respective decision stump. We consider the employment of a subset of features in a good decision stump as an indication of the subset being promising and we score it accordingly. The features of the dataset are then reordered based on the yielded ranking and the user can choose a number of high-ranked features for the classification task. We also use

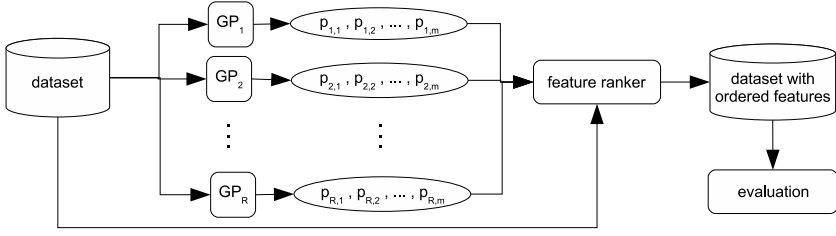


Fig. 1. Overview of the system

the projected dataset to evaluate the effectiveness of our system by feeding the highly ranked features to different classifiers and analysing their performance and behaviour.

2.1 Constructing Decision Stumps Using GP

A dataset D for a classification task is defined to be a set of pairs of input vectors $\mathbf{x}$ and class labels c :

$$D = \{(\mathbf{x}, c) : \mathbf{x} \in X, c \in C\} \quad (1)$$

where $X \subseteq \mathbb{R}^n$, n is the number of features in the input space, $C = \{l_1, l_2, \dots, l_m\}$ and m is the number of distinct classes in the problem. Given a program p_l constructed for class l , the program generates a mapping $p_l : X \mapsto Y$ which transforms the multi-dimensional input space X to a one dimensional coordinate Y . This transformation makes a new transformed dataset D'_l :

$$D'_l = \{(\mathbf{y}, c) : \mathbf{y} \in Y, l \in C\} \quad \text{where} \quad Y = p_l(X) \quad (2)$$

We then construct a decision stump for this new dataset by finding a continuous interval I_l in Y that contains most of the instances of class l with a minimum number of instances from other classes. In practice we first find a continuous interval which covers all the instances of class c and then we exclude some outliers to diminish the effect of noisy observations. The decision stump will classify any $\mathbf{x}$ as an l if and only if $y = p_l(\mathbf{x})$ falls in the interval.

An interval is discriminative if it does not contain a lot of instances from other classes. We treat the discrimination power of an interval as the fitness of the respective GP program. We measure the goodness of an interval by constructing a fitness function based on the fraction of instances of other classes that are correctly rejected:

$$n(I_l) = \frac{NTN}{NNL} \quad (3)$$

where NTN is the number of true negatives (correctly rejected) and NNL is the number of instances not in class l .

Fitness function. The steps involved in determining the fitness of a GP program p_l for a target class l (a decision stump for discriminating the instances from class l) over a training set D are as follows:

1. Use program p_l to convert each example in the training set D to a new transformed space D' . The program uses the feature values from the dataset and produces a single floating point value for each instance. So D' is a new dataset with a single one-dimensional attribute Y and a single class label for each instance.
2. Find a continuous interval I_l in Y which contains 99% of the instances from class l (i.e. excluding the first half of the data points before the first percentile and the second half of the data points after the last percentile).
3. Calculate the normalised fitness according to equation 3.

Each GP run generates m programs (one for each class). If the system performs R runs, it will generate a total of $R \times m$ GP programs where m is the number of distinct classes in the problem. These programs along with their fitness will be sent to the next phase to evaluate the rank of the features used.

2.2 Feature Ranking

GP is able to implicitly choose a good subset of features during the course of evolution to satisfy an objective function. However, GP cannot provide any explicit ranking or measure of goodness for the features. Here, we trace the other available information provided by GP back to the quality of the features used. The information we use includes the fitness of generated programs and the features they have used.

Each feature in the problem is credited if it has been used in a well constructed decision stump. The score that each feature receives for its appearance in a GP program is proportional to the fitness of that program. Features accumulate the scores they receive. So the features which are used more frequently in the GP programs gain more score. The local score gained by a feature f with respect to a GP program p is defined as:

$$s(f, p) = \begin{cases} \frac{n(p)}{|F_p|}, & f \in F_p \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

where F_p is the set of features used in program p , and $n(p)$ is defined by equation 3. Effectively, this formula divides the fitness of a program equally among the features used in the program. The global score of feature f can be obtained by summing the local scores over all programs and normalising:

$$S(f) = \frac{\sum_{i=1}^R \sum_{j=1}^m s(f, p_{i,j})}{\sum_{g \in F} \sum_{i=1}^R \sum_{j=1}^m s(g, p_{i,j})} \quad (5)$$

where R is the number of GP runs, m is the number of distinct classes in the problem, $p_{i,j}$ is the best program in the i th run for the j th class in the problem, and F is the set of all features. The denominator normalises the final score

relative to the total score gained by all the features. Finally, the features are ranked based on their score. The feature with the highest score is considered the best and is ranked first.

3 Experiment Design

3.1 Evaluation

We have designed a set of experiments to evaluate the effectiveness of our proposed GP-based feature ranking method. Since there is no direct way to measure the performance of a feature ranking/selection system, we evaluate our system by analysing changes in classification performance caused by using highly ranked features. In particular, we are interested in knowing if we can have a high performance by using just a few highly ranked features. We expect a quick increase in the classification performance by using highly ranked features. In contrast we do not expect low ranked features to have a considerable effect on increasing the performance.

3.2 Datasets

We used three datasets with a relatively large number of features from the UCI machine learning repository [14] in the experiments. Table 1 summarises the main characteristics of these datasets.

Table 1. Datasets

Problem	# Features	# Nominal	# Instances	# Classes
Johns Hopkins Ionosphere	34	0	351	2
Wisconsin Breast Cancer Diagnostic (WBCD)	30	0	569	2
Sonar	60	0	208	2

3.3 Settings and Implementation Details

We use tree-based genetic programs, each of which produces a single floating point value as output. There is one variable terminal for each feature in the problem. A number of randomly generated constants are also used as terminals. The function set consists of four elementary binary arithmetic operators: $+$, $-$, $\times$, $\div$. The $\div$ operator is “protected” which means it returns zero for division by zero. The ramped half-and-half method [15] is used for generating programs in the initial population and for the mutation operator. An elitist policy is taken by keeping the best individual of the generation. The population size is 1024. The initial maximum program tree depth is set to 4, but it can be increased to 8 during the evolution. The evolution is terminated, at the latest, after the 50th generation or when the fitness is maximised. To have enough GP programs to

use in the feature ranking phase, the number of GP runs is set to 300 and because all the tasks are binary classification, the total number of best constructed programs used in feature ranking is 600. The platform is implemented in Java and we use grid computing to have parallel GP runs.

Four types of classifiers are used in our experiments, namely the J48 implementation of C4.5 decision tree [16,17], Bayesian networks, Naïve Bayes [18], and the SMO version of the SVM classifier [19]. All the performance measures are evaluated via 10-fold cross-validation. We use the Weka [17] library for the classification and evaluation processes.

4 Results and Discussion

4.1 Feature Ranking Results

The first stage generates scores for the features of the three problems; bar charts of the scores are shown in Figures 2 and 3. In each chart, the horizontal axis shows the feature index, starting from 1, and the vertical axis shows the scores (computed by equation 5) gained by the features. Note that the scores are relative and the absolute values are not significant. Table 2 shows the rank of the features in each classification task. The features are listed in the order of importance starting with the most important one.

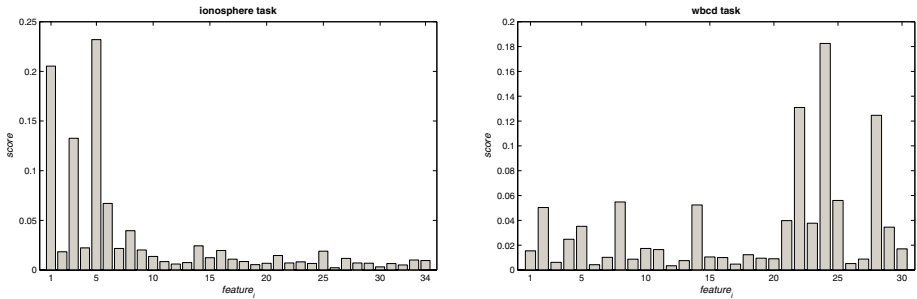


Fig. 2. Score of features in the John Hopkins ionosphere dataset (left) and the Wisconsin breast cancer dataset (right)

4.2 Classification Performance with Ranked Features

The second stage used the ranked features for classification. For each task, we measured the performance of the four classifiers with different sets of features using 10-fold cross-validation. The classifiers were first trained with just the highest ranked feature, then with two highest ranked features, then with three highest ranked features and so on up to all available features. The results which are the average over 30 10-fold cross-validations are plotted in Figures 4, 5 and 6. In each

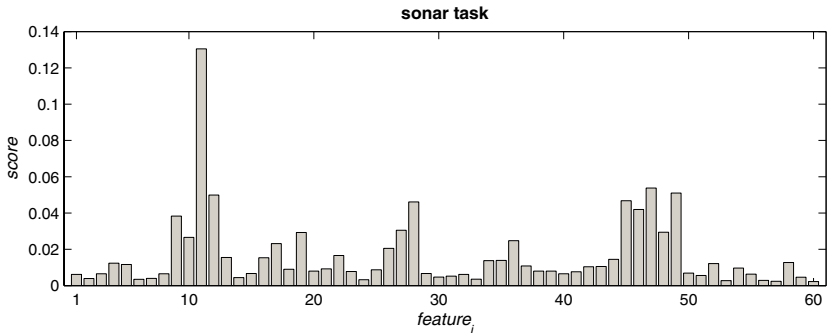


Fig. 3. Score of features in the sonar dataset

Table 2. Feature ranks

Problem	Order of features
Johns Hopkins Ionosphere	5, 1, 3, 6, 8, 14, 4, 7, 9, 16, 25, 2, 21, 10, 15, 27, 17, 33, 34, 18, 11, 23, 13, 22, 28, 29, 20, 24, 31, 12, 19, 32, 30, 26
Wisconsin Breast Cancer Diagnostic	24, 22, 28, 25, 8, 14, 2, 21, 23, 5, 29, 4, 10, 30, 11, 1, 18, 15, 7, 16, 19, 20, 27, 9, 13, 3, 26, 17, 6, 12
Sonar Dataset	11, 47, 49, 12, 45, 28, 46, 9, 27, 48, 19, 10, 36, 17, 26, 22, 13, 16, 44, 35, 34, 58, 4, 52, 5, 37, 43, 42, 54, 21, 18, 25, 38, 39, 20, 23, 41, 50, 15, 29, 8, 40, 3, 55, 32, 1, 51, 31, 30, 59, 14, 7, 2, 33, 6, 24, 56, 53, 57, 60

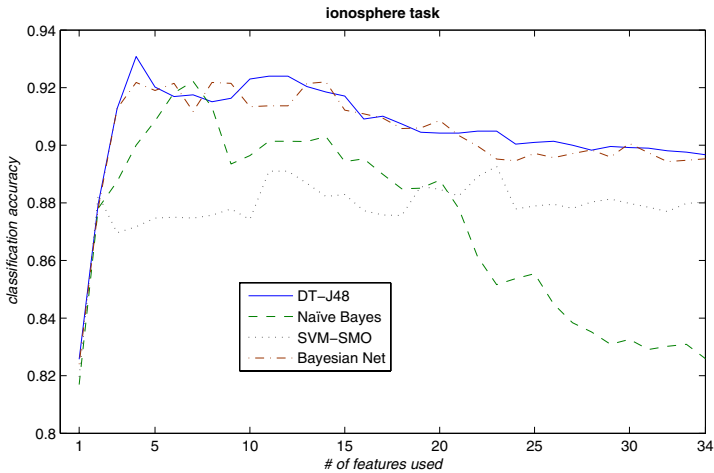


Fig. 4. Accuracy of different classifiers in the John Hopkins University ionosphere classification task by using different number of ranked features

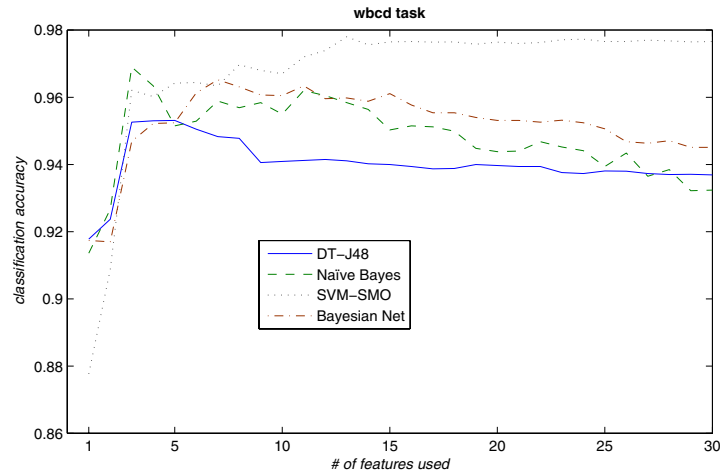


Fig. 5. Accuracy of different classifiers in the Wisconsin breast cancer classification task by using different number of ranked features

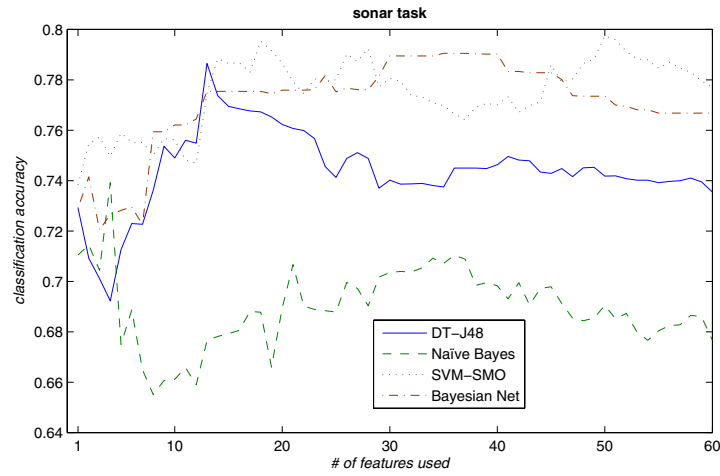


Fig. 6. Accuracy of different classifiers in the sonar classification task by using different number of ranked features

figure, the horizontal axis shows the number of features used in the classification and the vertical axis in each figure shows the classification performance.

Considering the performance figures of the three classification tasks, we observe that classification performance increases very quickly as the first few highly ranked features are added. In the ionosphere task, compared to the situation where all the 34 features are used, naïve Bayes and SVM can do better by using only 2 features, and DT and Bayesian Net can do better by using just 3 features.

It is almost the same in the WBCD task where for all classifiers except SVM the performance that is obtained by using all features can be obtained by using less than 4 features out of 30. In the sonar task, DT, Naïve Bayes, SVM and Bayesian network can, by using 8, 1, 14 and 13 feature(s) respectively, perform better than situations in which all 60 features are used. For almost all the classifiers, by using less than 10% of the features, we can obtain a similar performance or better than that obtained by using all the features.

Looking at the trends in the classification performance, it is revealed that in most cases, using more features with these classifiers, not only does not increase the performance, but actually causes a considerable deterioration. This is particularly true for all the classifiers in the ionosphere task and all the classifiers in the WBCD task except SVM. This is probably due to the fact that low ranked features are actually noisy and do not contribute towards discriminating class instances. Ideally we expect a good learning algorithm to perform monotonically better by feeding more information to it. However, in practice many well-known machine learning techniques are severely sensitive to noisy data. Therefore limiting the dataset to just highly ranked features seems essential.

Using all the features does not achieve the best result in any of these three tasks. By using the ranked features incrementally, we can find some subsets of features that lead to better classification performance in each problem. If we think of the number of features used in the performance figures as a linear search space of subsets of features, there will be one subset with cardinality of 1 (containing the highest ranked feature), one subset with cardinality of two (containing the two highest ranked features) and so on. While the size of the actual search space is 2^n where n is the total number of features in the task, the size of this shrunken linear search space is just n . However, looking at the performance graphs for all the tasks and all the classifiers, we observe that there is a point in this shrunken search space where the classification performance is better than the performance obtained when all the features are used. The shrunken search space is so small that it can be exhaustively searched in $O(n)$.

5 Conclusions and Future Work

While most feature ranking methods try to find a direct relation between one single feature and the target class label to rank features, we use GP to rank features based on their appearance in groups of features and the usefulness of the groups in discriminating class labels. GP is a good method for finding the hidden relationship between a subset of features and the target class label. It also has an intrinsic capability in selecting good features to satisfy an objective function. However, as the goodness of selected features can not be explicitly quantified by GP, we provide a scoring mechanism in parallel with the GP search.

We find that, in most cases, by using less than 10% of the highly ranked features, we can gain the same classification performance as that gained when all the available features are used. The proposed ranking system is not only good for finding good features, but it is also useful in detecting noisy and useless

features. As we see in the results, adding too many low-ranked features to most of the classifiers causes considerable deterioration in the classification performance. GP-based feature ranking also provides a shrunken search space (of the subsets of features) in which there is at least one point which yields a better classification performance for any type of classifier.

As future work, one can investigate whether the high-ranked features are correlated or redundant and if so, how. It would also be of interest to find an effective way to investigate if there exists a combination of high- and low-ranked features that can improve the classification performance.

References

1. Jong, K., Mary, J., Cornujols, A., Marchiori, E., Sebag, M.: Ensemble feature ranking. In: Boulicaut, J.-F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) PKDD 2004. LNCS, vol. 3202, pp. 267–278. Springer, Heidelberg (2004)
2. Ruiz, R., Riquelme, J.C., Aguilar-Ruiz, J.S.: Fast feature ranking algorithm. *Knowledge-Based Intelligent Information and Engineering Systems*, 325–331 (2003)
3. Biesiada, J., Duch, W., Kachel, A., Maczka, K., Palucha, S.: Feature ranking methods based on information entropy with parzen windows. In: *International Conference on Research in Electrotechnology and Applied Informatics (REI 2005)*, pp. 109–119 (2005)
4. Lin, T.H., Chiu, S.H., Tsai, K.C.: Supervised feature ranking using a genetic algorithm optimized artificial neural network. *Journal of Chemical Information and Modeling* 46, 1604–1614 (2006)
5. Oh, I.S., Lee, J.S., Moon, B.R.: Hybrid genetic algorithms for feature selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1424–1437 (2004)
6. Smith, M.G., Bull, L.: Genetic programming with a genetic algorithm for feature construction and selection. *Genetic Programming and Evolvable Machines* 6, 265–281 (2005); Published online: 17 August 2005
7. Langdon, W.B., Poli, R.: *Foundations of Genetic Programming*. Springer, Heidelberg (2002)
8. Lin, J.Y., Ke, H.R., Chien, B.C., Yang, W.P.: Classifier design with feature selection and feature extraction using layered genetic programming. *Expert Syst. Appl.* 34, 1384–1393 (2008)
9. Davis, R.A., Charlton, A.J., Oehlschlager, S., Wilson, J.C.: Novel feature selection method for genetic programming using metabolomic 1h nmr data. *Chemometrics and Intelligent Laboratory Systems* 81, 50–59 (2006)
10. Liu, H., Yu, L.: Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering* 17, 491–502 (2005)
11. Landry, J.A., Costa, L.D., Bernier, T.: Discriminant feature selection by genetic programming: Towards a domain independent multi-class object detection system. *Systemics, Cybernetics and Informatics* 3, 76–81 (2006)
12. Parrott, D., Li, X., Ciesielski, V.: Multi-objective techniques in genetic programming for evolving classifiers. In: *The 2005 IEEE Congress on Evolutionary Computation*, 2005, vol. 2, pp. 1141–1148 (2005)

13. Muni, D.P., Pal, N.R., Das, J.: Genetic programming for simultaneous feature selection and classifier design. *IEEE Transactions on Systems, Man and Cybernetics, Part B* 36, 106–117 (2006)
14. Asuncion, A., Newman, D.: UCI machine learning repository (2007), <http://www.ics.uci.edu/~mllearn/MLRepository.html>
15. Koza, J.R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge (1992)
16. Quinlan, J.R.: *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Francisco (1993)
17. Witten, I.H., Frank, E.: *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd edn. Morgan Kaufmann, San Francisco (2005)
18. John, G., Langley, P.: Estimating continuous distributions in bayesian classifiers, pp. 338–345 (1995)
19. Keerthi, S.S., Shevade, S.K., Bhattacharyya, C., Murthy, K.R.K.: Improvements to platt's smo algorithm for svm classifier design. *Neural Comp.* 13, 637–649 (2001)

Time Series Prediction with Evolved, Composite Echo State Networks

Russell Y. Webb

University of Canterbury, Christchurch 8001, NZ

r.webb@elec.canterbury.ac.nz

<http://www.elec.canterbury.ac.nz/people/webb.shtml>

Abstract. A framework for predictive, on-line, learning networks composed of multiple echo state networks is presented. These composite networks permit learning predictions based on complex combinations of sub-predictions and error terms. The configuration space is explored with a genetic algorithm and better performance is achieved than with hand coded solutions.

1 Previous Work

In previous research by the author [1], a new cascade, error-correction network (CECN) was investigated. A cascade of echo state networks (ESNs) [2,3] was trained on-line such that the first predicted the value of a chaotic time series [4] (the Mackey-Glass [5] oscillator), and each subsequent ESN predicted an error correction for the previous ESN. Each ESN had one input and was trained to predict that input (typically one time step into the future). Once trained, a summation over all the ESN outputs was shown to significantly improve prediction accuracy and variability when compared to a number of other topologies using the same neural resources.

Aspects of the previous cascade architecture which are refined in the present work include the following:

- Ring versus random connections within the ESN.
- Method for creating outputs from a multi-prediction architecture.
- Generalised ESN network topology.

The ESN simulations used here are largely identical to those from the previous work. For completeness, the major settings used for ESN configuration are listed below.

Training steps	8,000	Testing steps	2,000
Number reservoir neurons	40	Spectral radius	0.95
Activation function	<i>tanh</i>	Learning rate, γ	0.20
Prediction time horizon	1	Predictions per ESN	2

The update and learning equations for a each prediction are as follows, using y for input and z for training signals. The output from the prediction is a estimate of the future value and an error of the last prediction, $(\hat{z}_{t+1}, \delta_t)$.

$$N = \tanh(MN + W_{in}y_{1,t} + W_{in}y_{2,t}) \quad (1)$$

$$S_t = [N, y_{1,t}, y_{2,t}] \quad (2)$$

$$\hat{z}_{i,t+1} = \tanh(S_t W_{out,i}) \quad (3)$$

$$W_{out,i} = W_{out,i} + S\gamma_t(\operatorname{atanh}(z_{i,t+h}) - \operatorname{atanh}(\hat{z}_{i,t+1})) \quad (4)$$

$$\delta_{i,t} = z_{i,t} - \hat{z}_{i,t}. \quad (5)$$

1.1 Adding Input and Output Ports

The present work emerged from asking the simple question, how can multiple ESNs which each have two input and two training ports be optimised for prediction? Such a composite network of k ESN nodes will be referred to as a k -net. Manual configuration quickly became unmanageable, as even the optimal 1-net is non-obvious, and few 2-nets could be intuitively ruled out.

A detailed schematic representation of an ESN predictor is shown in Figure-1. The important differences between the previous CECN and the current system are listed below:

- Implemented in C++ rather than Matlab.
- Two input signals per ESN.
- Two predictions and error terms are formed by each ESN.
- The predicted signals can be any signal in the network (rather than being restricted to predicting one of the inputs to the ESN).

1.2 Confirmation of Previous Results

The design goal is to optimally predict a time series using a k -net. Online training is used for simplicity, scalability, and plausible, localised learning. Since the previous simulation was written in Matlab and the new simulation is written in C++, the first goal was to replicate the previous CECN results. By specifying the network connections, the 3-net shown in Figure-2 was created. Note, ports receiving input from the time series are circled in black, and some ports are not connected (they receive an input of 0).

Averaging the results over 100 runs gave an NSME value of 5×10^{-3} , which is within a factor of 3 of the previous result. Connecting the ESN neurons into a ring with uniform connection strength of 0.95, yields an average NMSE of 1.3×10^{-3} , which is almost exactly the result found previously in the Matlab CECN simulation. This agreement hints that the method of spectral radius scaling for random connection matrices may be the primary difference between the simulations. Both spiral [6] and ring [1,7] connectivity within the ESN reservoir are known to yield reliable and accurate results in some applications.

Ring connectivity is useful for a number reasons. First, it is faster to simulate and easier to implement in hardware. Second, ring ESNs are much less random than conventional ESNs; in a conventional ESN, there is always a chance for random wiring to lead to poor results. The small population, genetic algorithm

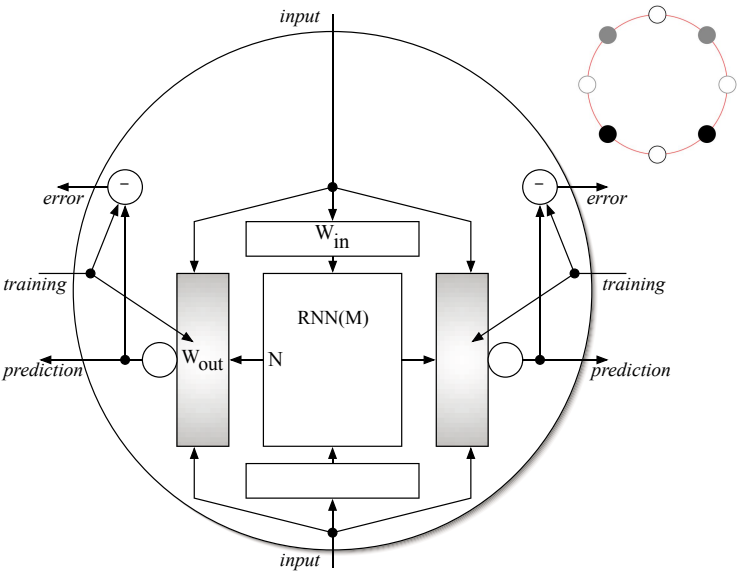


Fig. 1. ESN predictive node predicting two time series from two inputs. The abstract symbol for a node is shown in the upper-right in the same orientation; the input and output ports are shown as open and filled circles respectively.

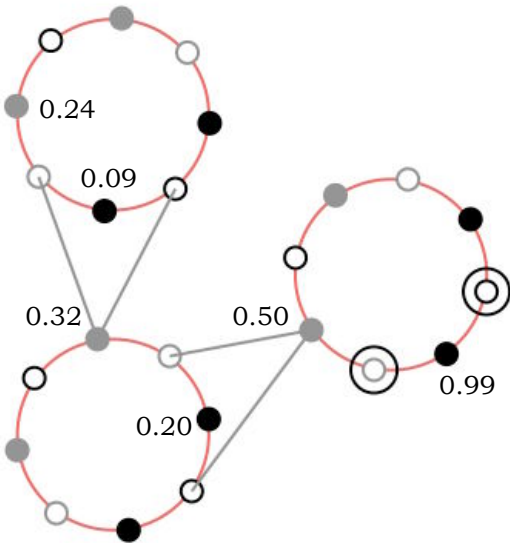


Fig. 2. Error correction cascade with learned output filter

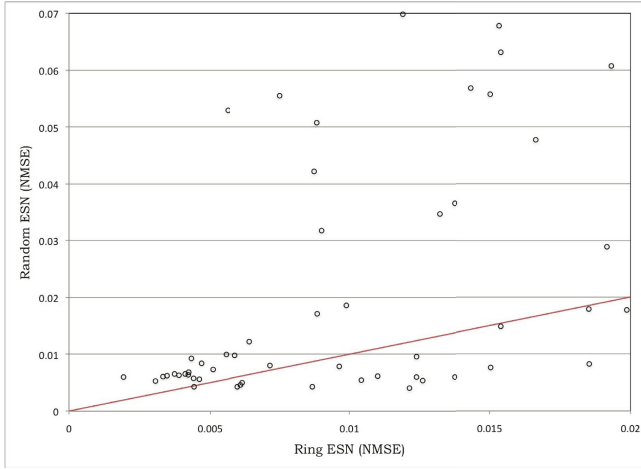


Fig. 3. Comparison of errors from ring and random ESN based networks with the same connections. Points on the line have the same performance in either configuration.

used to optimise connectivity will benefit from this stability. Finally, for the time-series prediction tasks used here, ring ESNs seem to work well.

Figure-3 shows the best of a thousand random 3-nets simulated with random and ring connectivity on the M-G oscillator prediction task. For the best networks sampled, random connections and ring connections have nearly equal performance. To make the ring ESN even more predictable, the input vectors W_{in} are no longer random. Instead, each input weight vector is a linear ramp beginning at zero and increasing in opposite directions around the ring.

1.3 Output Filter Improvement on CECN

The most interesting side-effect of evolved or random network connectivity is that it is no longer clear how to form a prediction from the available signals. In the CECN, each output was a correction term relative to the previous, so a simple summation produced a viable prediction. For k -nets, a linear filter was implemented to combine all available signals into a prediction. This output filter is learned on-line, in-parallel with the predictor. The final prediction from the k -net is $\hat{x}_{t+1} = \sum_{i=1}^{4k} f_i s_i$ where f is a value in the output filter and s is a signal produced by one of the nodes.

As seen in the coefficients shown in Figure-2, when simulating a CECN the result is not a simple summation of predictions. The learned weighted sum roughly halves the error of the network given by a summation of the error corrections. This improvement results from the filter using signals which are not strictly synchronised in time with the desired prediction.

These filter weights will vary based on the nature of the signal to be predicted. For instance, continuous signals will allow the k -net to combine values in the network which are referring to time steps other than the one being predicted;

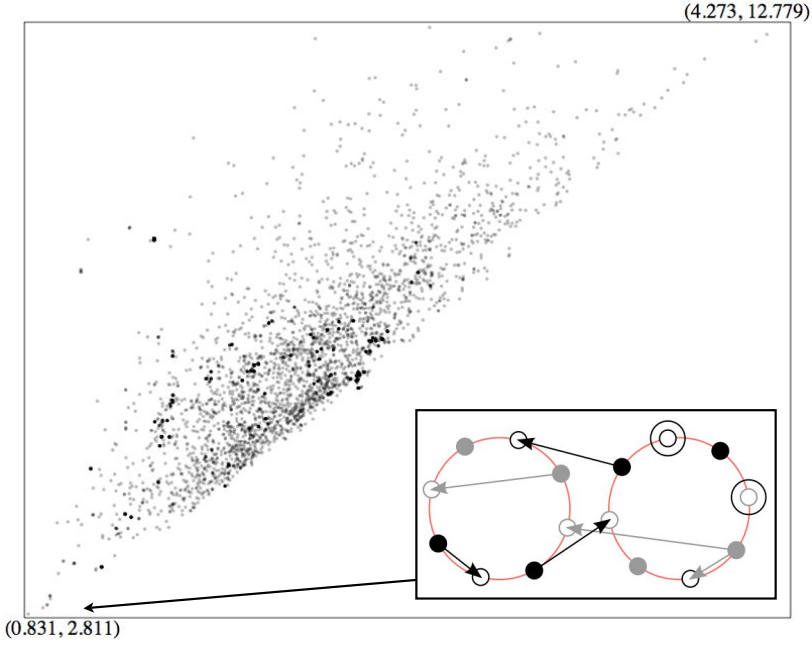


Fig. 4. Maximum filter separation (x-axis) versus combined error, err , (y-axis) for 7,100 random 2-nets

while, spiky signals will use filters that do not mix values over time. To gain further insight, a range of signals were presented to randomly generated 2-net configurations. The five series used were both variables of the M-G oscillator, the logistic map, $\sin(f_1 t)$, and $\sin(f_1 t) + \sin(f_2 t)$ with non-harmonic frequencies. The logistic map is the most difficult of these series with a NMSE of 0.024 being achieved by a 3-net.

A large separation in the output filter weights between different time series means that the filter strongly shapes the result, while a distance of zero means there is a fixed way of combining the network signals to produce a prediction. Figure-4 compares the maximum distance, $dist$, between two filters any two filters and the magnitude of the combined normalised-errors, err , of all five series:

$$err = \sqrt{\sum_{i=1}^5 \left(\frac{NMSE_i}{\sigma_i} \right)^2}. \quad (6)$$

This result indicates that the best performing filters, in terms of prediction accuracy (err), are also the most general (they use similar output filters for different signals). The best of these 2-nets is shown in the Figure-4 inset. If we define each node to be a operator taking two input values and producing two

outputs composed of a prediction and an error, then k -nets can be described using the following notation:

$$\dot{z} \equiv (z_p, z_e) = (\hat{z}_{t+1}, z_t - \hat{z}_t) \quad (7)$$

$$\dot{z}_1, \dot{z}_2 = \mathcal{P}(y_1, y_2, | z_1, z_2). \quad (8)$$

The 2-net from the inset can be specified recursively by introducing new names on the left-hand side:

$$a, b = \mathcal{P}(x, a_e, | x, d_p) \quad (9)$$

$$c, d = \mathcal{P}(c_p, b_p, | d_e, a_e). \quad (10)$$

The average of the output filters for this predictor over the sequences gives a final prediction,

$$\hat{x}_{t+1} = 0.95a_p + 0.33a_e + 0.27c_p + 0.30d_e, \quad (11)$$

$$\hat{x}_{t+1} \sim 0.95x_p + 0.33x_e + 0.27x_{eep} + 0.30x_{ee}. \quad (12)$$

The composition of operations has been denoted as concatenation in the subscript, such that z_{ep} is a prediction, p , of an prediction-error, e , of z . The final form is only a similar expression because it omits all information about connections and describes all data in relation to the target series, x . For each signal, the number of p operations in the subscript indicates how many steps into the future the signal is about. A diagram which simplifies the tracking of information in the network is presented in Figure-5. Even though the four terms have zero weight, removing any connection degrades the accuracy of the network. The present work is focused on finding optimal configurations, however analysis is under way to understand the information flow in these recurrent predictor networks.

2 Search for Optimal Solutions

2.1 Size of Search Space

Using Burnside's Theorem [8], the number of unique network configurations under symmetry transformations can be calculated by taking the total number of configurations and dividing by the size of the symmetry group, S_{net} :

$$N_{unique} = \frac{N_{total}}{|S_{net}|} \quad (13)$$

The total number of configurations is the number-of-output-nodes raised to the number-of-input-nodes power. For a network with k nodes, that formula is $N_{unique,k} = (4k + 2)^{4k}$. The two extra outputs come from the time series and null values signals.

The symmetry group for a k -net is less obvious. Each node has the same symmetry group as the rectangle (identity, two reflections, and rotation by 180 degrees). For a 1-net, the size of the symmetry group is 4. A 2-net has the

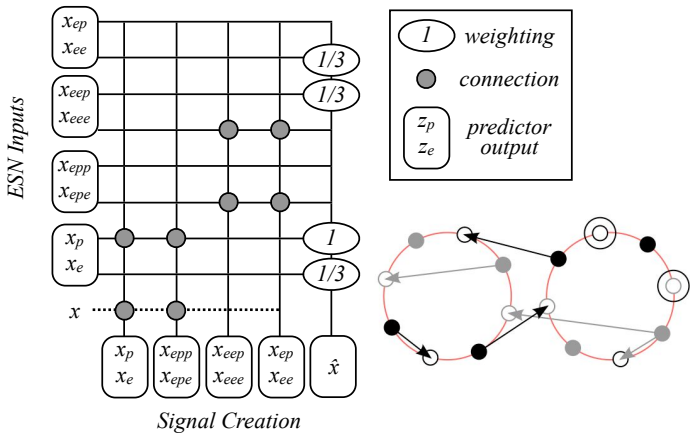


Fig. 5. A table based notation for the inset network showing which inputs are used to calculate each signal, how those signals are reused, and how they are combined to form the final prediction

additional symmetry which allows the two nodes to be swapped, so a 2-net has a $2 \times 4 = 8$ symmetry operations (each node can be operated on in either location and the locations can be swapped). The 3-net is the largest net studied here and has symmetries formed by the product of the triangle symmetry group and the node group: $6 \times 4 = 24$.

The number of unique configurations for each size network studied is shown below:

1-net	$((4 + 2)^4)/4$	324
2-net	$((8 + 2)^8)/8$	12,500,000
3-net	$((12 + 2)^{12})/24$	2.36×10^{12}

If 100 configurations can be simulated per second, the 3-net space will take nearly 750 years to search (neglecting other tuneable parameters such as number of ESN neurons, spectral radius, learning rate, and sparseness). Note, any asymmetry between the two inputs into a node will double number of unique configurations.

2.2 GA Representation and Operations

In order to address the large search space, a genetic algorithm (GA) was implemented to optimise connections between nodes. Here the GA operates on connections external to the ESN nodes, rather than on the ESN itself [10]. Many fascinating studies have applied GAs to neural computing; an excellent review of neuroevolution was published by Floreano [11].

The results presented here were obtained from single population evolution, minimising the prediction error as fitness. Diversity pressure was achieved by randomly selecting among the five time series discussed in Section-1.3 and by

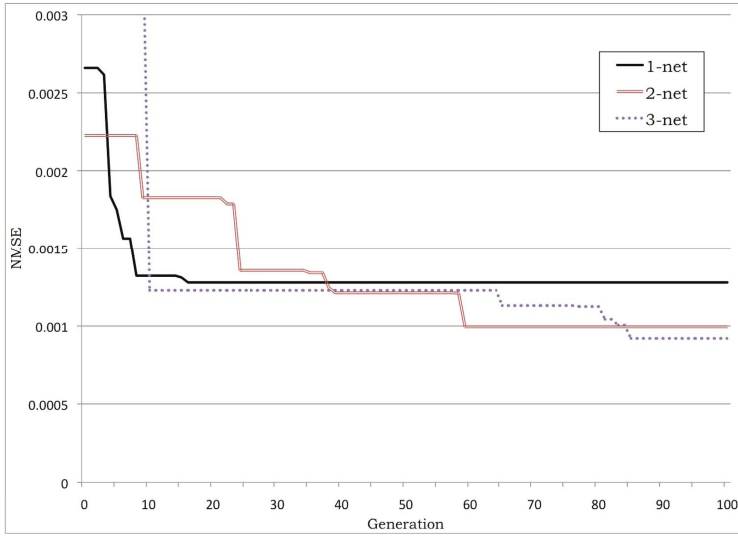


Fig. 6. Best fitness as solutions evolve for networks with different number of nodes

free selection of the second parent for hybridisation. The most recent winner for each of the five series was protected from replacement in order to maintain the best solutions in the population. A population of 46 individuals was used, and the lower half was replaced in each generation via either a new random network (20%), a mutation of a solution from the top half of the population (40%), or a hybrid of one top individual and an individual selected from the whole population (40%). Uniform selection was used within those groups.

2.3 Solutions Found

A trace of the fittest solution is shown in Figure-6 for 1, 2, and 3-nets. As expected, larger networks have initially higher errors but eventually find better solutions. The 1-net in particular does not seem to make further progress after generation 17; this solution has been confirmed to be optimal via exhaustive search of 1-net configurations.

The best solution for the 1-net (Figure-7) provides insight into the type of results produced. This solution is a combination of the primary prediction and three corrective terms. This 1-net is performing error correction by predicting a correction term for the main prediction, but it also mixes in the time delayed error terms for the M-G prediction task.

The current best 2 and 3-net solutions are not believed to be optimal, but they both outperform the hand designed CECN (0.0013 NMSE). As seen in

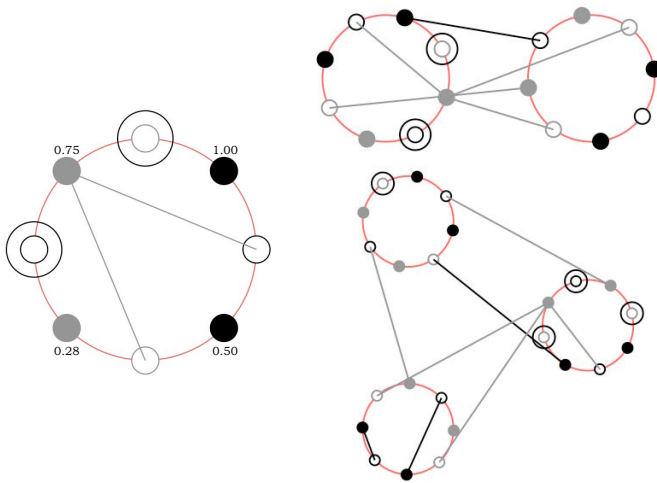


Fig. 7. Best 1, 2, and 3-net solutions found via GA search

Figure-7, the 3-net solution does not appear to adequately use the lower node (an observation confirmed by analysis of the importance of each connection).

3 Conclusions and Future Work

Generalised predictive ESN networks have been demonstrated and evolved into solutions that outperform more intuitive, designed solutions. The primary goal of understanding optimal learning on recurrent networks of prediction and error remains open. To that end, there are many areas still under investigation:

- Interaction of sub-populations.
- Cluster computing implementation.
- Larger networks.
- Improved normalisation in the random ESN implementation.
- Evolution of ESN internal parameters (including the input weight pattern).
- Spiral ESN connections.
- Analysis methods based on information flow.
- Classification of features within k -nets and high level models of performance based on features.

Acknowledgments

The author thanks the University of Canterbury and specifically the College of Engineering and Department of Electrical and Computer Engineering for support in conducting this research. Additional thanks goes to the many colleagues and students who inspired the work with insightful discussion.

References

1. Webb, R.Y.: Multi-layer corrective cascade architecture for on-line predictive echo state networks, *Applied Artificial Intelligence* (2008) [in publication]
2. Jaeger, H., Haas, H.: Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication. *Science* 304(5667), 78–80 (2004)
3. Schrauwen, B., Verstraeten, D.: An overview of reservoir computing: theory, applications and implementations. In: *Proceedings of the 15th European Symposium on Artificial Neural Networks*, pp. 471–482 (2007)
4. Kountouriotis, P.A., Obradovic, D., Goh, S.L., Mandic, D.P.: Multi-Step Forecasting Using Echo State Networks. In: *EUROCON 2005*, November 22–24 (2005)
5. Mackey, M.C., Glass, L.: Oscillation and chaos in physiological control systems. *Science*, 287–289 (1977)
6. Gao, H., Sollacher, R., Kriegel, H.P.: Spiral Recurrent Neural Network for Online Learning. In: *ESANN 2007*, Bruges, Belgium, April 25–27, 2007, pp. 483–488 (2007)
7. Verstraeten, D., Schrauwen, B., Campenhout, J.V.: Adapting reservoirs to get gaussian distributions. In: *Proceedings of the 15th European Symposium on Artificial Neural Networks 2007*, April 2007, pp. 495–500 (2007)
8. Shifrin, T.: *Abstract Algebra: A Geometric Approach*. Prentice Hall, Englewood Cliffs (1996)
9. Jiang, F., Berry, H., Schoenauer, M.: Supervised and Evolutionary Learning of Echo State Networks. In: *10th International Conference on Parallel Problem Solving From Nature* (2008)
10. Xu, D., Lan, J., Principe, J.C.: Direct Adaptive Control: An Echo State Network and Genetic Algorithm Approach. In: *Proceedings of the Int. Joint Conference on Neural Networks*, Montreal, Canada, July 31 - August 4, 2005, pp. 1483–1486 (2005)
11. Floreano, D., Dürr, P., Mattiussi, C.: Neuroevolution: from architectures to learning. *Evolutionary Intelligence* 1, 47–62 (2008)

Genetic Synthesis of Software Architecture

Outi Räihä¹, Kai Koskimies², and Erkki Mäkinen¹

¹ University of Tampere, Finland

² Tampere University of Technology, Finland
outi.raiha@uta.fi

Abstract. Design of software architecture is intellectually one of the most demanding tasks in software engineering. This paper proposes an approach to automatically synthesize software architecture using genetic algorithms. The technique applies architectural patterns for mutations and quality metrics for evaluation, producing a proposal for a software architecture on the basis of functional requirements given as a graph of functional responsibilities. Two quality attributes, modifiability and efficiency, are considered. The behavior of the genetic synthesis process is analyzed with respect to quality improvement speed, the effect of dynamic mutation, and the effect of quality attribute prioritization. Our tests show that it is possible to genetically synthesize architectures that achieve a high fitness value early on.

Keywords: architecture synthesis, genetic algorithm, search-based software engineering, software design.

1 Introduction

A persistent dream of software engineering is to be able to automatically produce software systems based on their requirements. While the synthesis of executable programs is in general beyond the limits of current technology, the automated derivation of architectural designs of software systems is conceivable. This is due to the fact that architectural design largely means the application of known standard solutions in a combination that optimizes the quality properties (like modifiability and efficiency) of the software system. These standard solutions are well documented as architectural styles [1] and design patterns [2]. In addition, architectural design is guided by general principles like decomposition and usage of interfaces. Here we call all these solutions jointly (*architectural*) *patterns*.

In this paper we study the application of genetic algorithms [3] to software architecture synthesis. Architectural patterns provide a natural interpretation for genetic operations: a mutation can be realized as either the application or removal of an architectural pattern, and crossover operation can be realized by merging two architectures without breaking existing pattern instances. Fitness function can be expressed in terms of quality metrics available in the literature. Evaluating an architecture is a multi-criteria problem, but in the present implementation we have decided to add the objective functions together using the weighted sum approach. This is done for making the implementation more efficient. Our focus is on developing the required

techniques for genetic architecture synthesis, and on the investigation of the overall behavior of the genetic architecture synthesis process. The proposed architecture is produced as a UML class diagram with (possibly stereotyped) classes, interfaces and their mutual dependencies.

The main contributions of this work are a setup for genetic pattern-based software architecture synthesis starting from abstract requirements, and experimental analysis of the behavior of the genetic synthesis process, especially with regard to the development of fitness values. The former includes an approach to represent functional requirements as a responsibility graph, techniques for representing architectural information as genes, for computing quality based fitness, and for architectural cross-over and dynamic pattern-based mutation, and a demonstration of the genetic synthesis using an exemplary set of architectural patterns. The latter includes an analysis of the quality improvement speed, the effect of dynamic mutations, and the effect of prioritized quality attributes.

2 Related Work

Although our work differs significantly from what has been previously done in the field of search-based software engineering, genetic algorithms have been used widely for software clustering and refactoring.

The goal of software clustering or module clustering is to find the best grouping of components to subsystems in an existing software system. The problem is to partition the graph so that the clusters represent meaningful subsystems.

The genetic algorithm presented by Clarke et al. [4] for the clustering problem is quite straightforward: the main challenge is to find a suitable encoding, after which traditional mutation and crossover operators are used.

Harman et al. [5] approach the clustering problem from a re-engineering point of view: after maintaining a system its modularization might not be as good as it was when it was taken to use.

Seng et al. [6] represent the system as a graph, where the nodes are either subsystems or classes, and edges represent containment relations (between subsystems or a subsystem and a class) or dependencies (between classes). In this application each gene represents a subsystem, and each subsystem is an element of the power set of classes.

Systems refactoring is a more delicate problem than module clustering. When refactoring a system, there is the risk of changing the behavior of a system by, e.g., moving methods from a subclass to an upper class [7]. Hence, the refactoring operations should always be designed so that no illegal solutions will be generated.

O’Keeffe and Ó Cinneide [8] define the refactoring problem as a combinatorial optimization problem: how to optimize the weighting of different software metrics in order to achieve refactorings that truly improve the system’s quality. Seng et al. [7] have a similar approach, as they attempt to improve the class structure of a system by moving attributes and methods and creating and collapsing classes. O’Keeffe and Ó Cinneide [9] have continued their research with the use of the representation and mutation and crossover operators introduced by Seng et al. [7].

Harman and Tratt [10] introduce a more user-centered method of applying refactoring. They offer the user the option to choose from several solutions produced by the search algorithm.

Amoui et al. [11] have applied genetic algorithms for finding the optimal sequence of design pattern transformations to increase the reusability of a software system. Our approach is similar to Amoui et al.'s work in that we use high-level structural units, patterns, as the basis of mutations in a genetic process. We have also applied their supergene idea, to be discussed in Section 3, as a starting point for representing the architecture. However, there are several differences.

First, we consider not only reusability (or modifiability) as the quality criteria, but in principle we are interested in the overall quality of the architecture. In this paper we focus on two quality attributes, efficiency and modifiability.

Second, we aim at the synthesis of the architecture starting from requirement-level information, rather than at improving an existing architecture. Third, we do not restrict to design patterns, but consider more generally various kinds of architectural solutions at different levels.

Our viewpoint is different from that of system clustering and refactoring. System clustering considers software architecture only from the decomposition perspective, and software refactoring aims at structural fine-tuning of software architecture, whereas our approach strives for automating the entire architecture design process.

3 Technique for Genetic Architecture Synthesis

3.1 Representing Functional Requirements

A major problem in automated software architecture synthesis is the representation of functional requirements. We have adopted here an approach where functional requirements are represented as a responsibility dependency graph, each node representing a responsibility, and each directed edge representing a dependency between the two responsibilities. Here a responsibility denotes an elementary task or duty that can be identified for the system by analyzing its functional requirements (e.g. use cases). A responsibility depends on another responsibility if the former needs the latter to fulfill its task. These responsibilities remain as elements of the architecture as they are assigned to interfaces and classes, although they carry no semantics as far as the architecture synthesis is concerned. The architecture produced by the genetic synthesis reflects functional requirements only to the extent the responsibilities have been identified.

To allow the evaluation of the quality (here efficiency and modifiability) of the architecture, the responsibilities can (but do not have to) be associated with values characterizing, e.g., parameter size, time consumption, and the variability factor of the responsibility. However, in this paper, we assume that the values can be derived from the requirements; the quality of the results of this technique depends on the accuracy of these values. The given values for the attributes are relative, rather than absolute.

In this work we have used an intelligent home system as a case study [12]. Such a system provides an infrastructure and interfaces for controlling various home devices, like lights, drapes, and media devices. A fragment of the responsibility dependency

graph of this system is depicted in Figure 1, where the dependencies between and names of responsibilities are shown, as well as property values for variability factor, parameter size and time consumption (in this order). For example, the `drapeState` responsibility is a data responsibility, marked with a thicker line. In the middle of the graph is the responsibility `CalculateOptimalDrape`, which has a variability of 3, as the optimal drape position can be computed differently in different types of homes. As it is a heavy operation, its parameter size and time consumption are also among the highest values of those shown here. Responsibilities with such high attribute values play an important role when constructing quality architecture. The entire responsibility set for this system contains 52 responsibilities and 90 dependencies between them.

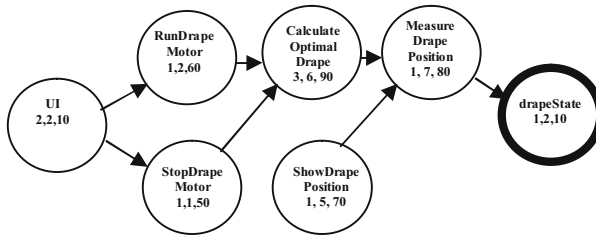


Fig. 1. Fragment of a responsibility dependency graph

3.2 Architectural Patterns

In the context of the present paper, an architectural pattern can be any general structural solution applied at the architectural level to improve some quality attribute of the system. Each architectural pattern gives rise to two mutation operations: introducing and removing the pattern.

In our experiments, we have used the following list of architectural patterns:

- decomposing a component
- using an interface
- Strategy design pattern [2]
- Façade design pattern [2]
- message dispatcher architectural style [1]
- communication through a dispatcher.

This collection of architectural patterns is of course very small, and intended only for experimentation purposes. We wanted to cover different levels of architectural patterns: basic practices, low-level design patterns (Strategy), medium-level design patterns (Façade), and high-level architectural styles (message dispatcher). The last architectural pattern is introduced for allowing components to join a message dispatcher introduced earlier. We expect that a real architecture synthesis tool would employ hundreds of architectural patterns.

3.3 Genetic Encoding of Software Architecture

In order for the genetic algorithm to operate on software architecture, the architecture needs to be represented as a chromosome consisting of genes. For efficiency, in this

experiment the architecture encoding is designed to suit the chosen set of architectural patterns. We have followed the supergene idea, introduced by Amoui et al. [11], where each gene has several fields to store data in. Taking this idea as a starting point, it is quite straightforward to place all information regarding one responsibility into one supergene. This also makes it easier to keep the architecture consistent, as no responsibility can be left out of the architecture at any point, and there is no risk of breaking the dependencies between responsibilities.

There are two kinds of data regarding each responsibility r_i . Firstly, there is the basic information concerning r_i given as input, containing the responsibilities $R_i = \{r_k, r_{k+1}, \dots, r_m\}$ depending on r_i , its name n_i , type d_i , frequency f_i , parameter size p_i , execution time t_i , call cost c_i and variability v_i . Secondly, there is the information regarding the responsibility r_i 's place in the architecture: the class(es) $C_i = \{C_{i1}, C_{i2}, \dots, C_{iv}\}$ it belongs to, the interface I_i it implements, the dispatcher D_i it uses, the responsibilities $RD_i \subset R_i$ that call it through the dispatcher, and the design pattern P_i it is a part of. The dispatcher is given a separate field as opposed to other patterns for efficiency reasons. Figure 2 depicts the structure of a supergene. The actual chromosome is formed by simply collecting all supergenes [12].

R_i	n_i	d_i	f_i	p_i	t_i	c_i	v_i	C_i	I_i	D_i	RD_i	P_i

Fig. 2. Supergene SG_i for responsibility r_i

Although basic operations in the architecture are relatively safe with this representation method (i.e., the responsibilities and their dependencies stay intact in the architecture), the design patterns produce challenges at the chromosome level, as careless operations can easily break patterns and make the architecture incoherent. Thus, in order to quickly check the legality of an operation with regard to patterns, a Pattern field is located in every supergene. The Pattern field has as attributes the classes, responsibilities and the interfaces involved with that particular pattern.

An initial population is first produced, where only basic structures, such as class division and interfaces for the responsibilities are randomly chosen [12]. To ensure as wide a traverse through the search space as possible, four special cases are inserted: all responsibilities being in the same class, all responsibilities being in different classes, all responsibilities having their own interface, and all responsibilities being as much grouped to same interfaces as the class division allows.

3.4 Mutation and Crossover Operations

All mutations are implemented as either introducing or removing an architectural pattern, i.e., decomposition, interfaces, message dispatcher and design patterns. This ensures a free traversal through the search space, as moves that may have seemed good at one time can be cancelled later on.

All mutations except for introducing a message dispatcher or a design pattern operate purely at supergene level by changing the value of the corresponding field. Introducing a new dispatcher to the system, however, is achieved by adding a “dummy” gene with only the dispatcher field containing data. Introducing design patterns, on the other hand, operate at supergene level, but affect more than one gene.

The legality of a mutation is always checked before it is administered to the selected gene. For this purpose, “architectural laws” have been defined. In our experiments, we have used three kinds of laws. Firstly, these laws ensure uniform calls between two classes: a class can communicate with another class only in a single manner (e.g. through an interface or through a message dispatcher). Secondly, the laws state some ground rules about architecture design, for example, that a responsibility can appear at most once in an interface. Thirdly, the laws regulate the order of introduction. For instance, a dispatcher must be introduced to the system before responsibilities can use it for communication.

Mutations are given a certain probability with which they are applied. The roulette wheel method [3] is used for selecting a mutation. A “null” mutation is also possible, giving a chromosome the chance to stay intact into the next generation. In addition, to study the effect of favoring more fundamental solutions in early stages, dynamic mutation probabilities have been defined for a set of patterns (dispatcher, Façade and Strategy). After 1/3 of the generations have passed, the probability of introducing a dispatcher to a system is decreased, and the probability of introducing a Façade is increased respectively. After traversing through another 1/3 of generations, the same is done with Façade and Strategy. The hypothesis is that favoring fundamental solutions (like architectural styles) in the earlier stages of evolution leads to a stronger core architecture that can be more easily refined at later stages by lower-level solutions.

In our approach, the crossover operation is also seen as a type of mutation, and thus, it is also included in the “roulette wheel”. The crossover is implemented as a traditional one-point crossover with a corrective function. In the case of overlapping patterns, the left side of the offspring is always considered to be the valid one, and the right side of the crossover point is corrected so that the whole architecture is valid.

The crossover probability increases linearly in regard to how high the fitness of an individual is in the population, which causes the probabilities of mutations to decrease in order to fit the larger crossover “slice” to the “wheel”. Also, after crossover, the parents are kept in the population for selection. These actions favor strong individuals to be kept intact through generations.

The actual mutation and crossover points are selected randomly. However, we have taken advantage of the variability property of responsibilities with the strategy and dispatcher communication mutations. The chances of a gene being subjected to these mutations increase with respect to the variability value of the corresponding responsibility. This should favor highly variable responsibilities.

3.5 Fitness Function

The fitness function is based on widely used software metrics [13], most of which are from the metrics suite introduced by Chidamber and Kemerer [14]. These metrics have been used as a starting point for the fitness function, and have been further developed and grouped to achieve clear “sub-fitnesses” for modifiability and efficiency, both of which are measured with a positive and negative metric. The biggest modifications to the basic metrics include taking into account the effect of interfaces and the dispatcher architecture style. A complexity metric is added to penalize having many classes and interfaces as well as extremely large classes.

Dividing the fitness function into sub-functions answers the demands of the real world, as hardly any architecture can be optimized from all quality viewpoints. Thus, we can assign a bigger weight to the more desired quality aspect. When w_i is the weight for the respective sub-fitness sf_i , the fitness function $f(x)$ for chromosome x can be expressed as

$$f(x) = w_1 * sf_1 - w_2 * sf_2 + w_3 * sf_3 - w_4 * sf_4 - w_5 * sf_5.$$

Here, sf_1 measures positive modifiability, sf_2 negative modifiability, sf_3 positive efficiency, sf_4 negative efficiency and sf_5 complexity. The sub-fitness functions are defined as follows ($|X|$ denotes the cardinality of X):

$sf_1 = \text{interface implementors} + \text{calls to interfaces} + (\text{calls through dispatcher} * \sum (\text{variabilities of responsibilities called through dispatcher})) - \text{unused responsibilities in interfaces} * 10$,

$sf_2 = \text{calls between responsibilities in different classes}$,

$sf_3 = \sum (\text{dependingResponsibilities within same class} * \text{parameterSize} + \text{lusedResponsibilities in same class} * \text{parameterSize} + \text{dependingResponsibilities in same class} * \text{parameterSize}))$,

$sf_4 = \sum \text{ClassInstabilities} + \text{dispatcherCalls} * \sum \text{callCosts}$, and

$sf_5 = \text{classes} + \text{interfaces} + \text{BigClassPenalty}$.

The multiplier 10 in sf_1 notes that having unused responsibilities in an interface is almost an architecture law, and should be more heavily penalized.

Selection of individuals for the next population is made with a roulette wheel selection, where the size of each “slice” is linearly in proportion to how high the corresponding individual’s fitness is in the population. No individual can be selected more than once. Thus, the “slices” are adjusted after each selection to represent the differences between fitnesses of the remaining individuals.

4 Experiments

In this section we present the results from the preliminary experiments done with our approach, using the example system introduced in Section 3.1. The algorithm was implemented in Java 1.5, and one test run with a population size of 100 and 250 generations took approximately 90 seconds. All test runs were conducted with a fixed set of mutation probabilities, found after extensive testing. The calculated fitness value is the average of 10 best fitnesses in each generation. In all experiments, the actual y-value for the curve is achieved as the average value from five test runs. The average value is used after first ensuring a similar fitness curve for all test runs. Examples of the produced UML diagrams are presented by R  ih   [12].

We first examined the overall development of the fitness values over a high number of generations. As can be seen in Figure 3, depicting the evolvment of fitness values over 1000 generations, the fitness values achieve their highpoint after around 750 generations, and achieve quite high values already after 500 generations.

To analyze the effect of weighing one quality evaluator over another, we have extracted the separate sub-fitness curves for modifiability and efficiency in cases where one was weighted heavily over the other. These tests were made with a population

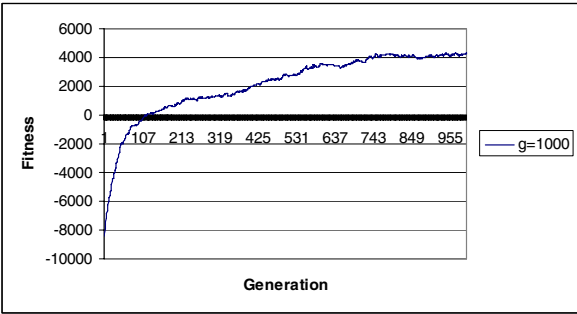


Fig. 3. Fitness development

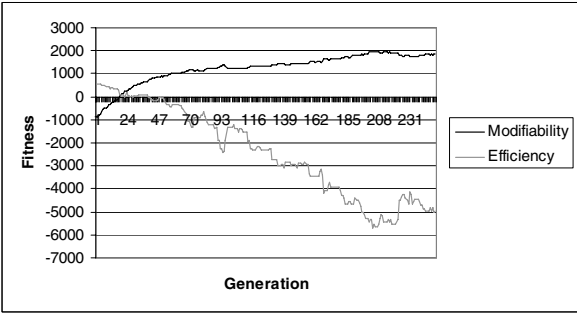


Fig. 4. Heavily weighted modifiability

size 100 and 250 generations. In the first test, depicted in Figure 4, the modifiability functions were weighted 10 times higher than the efficiency functions. This results in the “normal” development of the modifiability curve, while the efficiency curve plummets quite rapidly, and continues to worsen throughout the generations.

In the second test efficiency was correspondingly weighted 10 times higher than modifiability; the fitness curves are shown in Figure 5. In this case, the efficiency curve achieves very high values from the very beginning and does not develop as noticeably as the modifiability fitness in the previous case. The modifiability fitness, however, remains quite stable, achieving only low values. The explanation for the poor development of the efficiency curve lies within the special cases inserted in the initial population. As the efficiency fitness values big classes, it would assign a high fitness value for the case where all responsibilities are in the same class. From this initial case it is fairly easy to achieve individuals with very few classes and high efficiency fitness values from the very beginning.

Finally, the effect of dynamic mutation probabilities was analyzed by testing them against probabilities that remained the same throughout the generations; the curves for these tests are shown in Figure 6. As can be seen, with 250 generations and a population of 100, the fitness curve achieves its high point quite early when the mutations are static, but with the dynamic mutation probabilities, the fitness value continues to

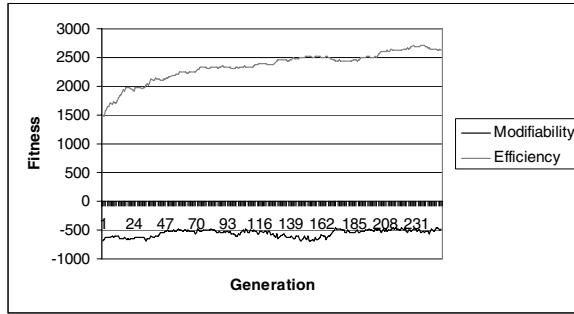


Fig. 5. Heavily weighted efficiency

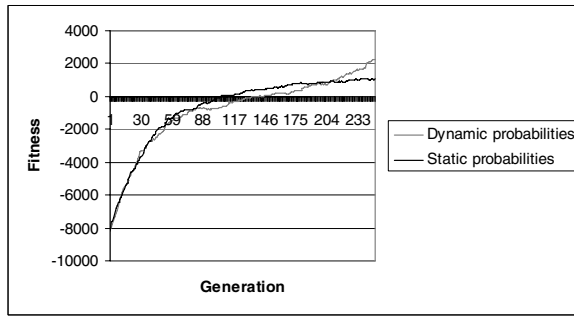


Fig. 6. Pattern probability variation

develop. Thus, it appears that dynamic mutation makes the basic structure of the architecture more amenable to fine-tuning in the later phases.

In this section we have shown that the quality of an architecture increases quite steadily with the selected evaluators related to modifiability, efficiency and complexity. If some quality attribute is heavily weighted in the process, it may have significant negative effect on another. Using dynamic mutation probabilities seems to offer advantages in longer generation sequences.

5 Concluding Remarks

We have presented a novel approach for genetic architecture synthesis. We have succeeded in genetically constructing an architecture from high-level responsibilities that achieves high quality values quite early in the development. The extremely rapid development of fitness values during the first 100 generations is especially notable, as it shows that a genetic algorithm can quickly find a “good” basic structure for software architecture. As the quality of software system is largely based on its architecture, our work also brings a new level to genetic programming. In genetic programming the emphasis has been to only produce programs that perform a certain task, while the quality factors of the produced code have so far been overlooked. Our plans for future

work include implementing a simulated annealing algorithm for comparison and more experiments on dynamic mutations. Moreover, we plan to implement a genuine multi-criteria approach, which would benefit in comparing the conflicting goals of modifiability and efficiency, discussed in Section 4.

References

1. Shaw, M., Garlan, D.: *Software Architecture - Perspectives on an Emerging Discipline*. Prentice-Hall, Englewood Cliffs (1996)
2. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading (1995)
3. Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolutionary Programs*. Springer, Heidelberg (1992)
4. Clarke, J., Dolado, J.J., Harman, M., Hierons, R.M., Jones, B., Lumkin, M., Mitchell, B., Mancoridis, S., Rees, K., Roper, M., Shepperd, M.: Reformulating software engineering as a search problem. *IEE Proceedings - Software* 150(3), 161–175 (2003)
5. Harman, M., Hierons, R., Proctor, M.: A new representation and crossover operator for search-based optimization of software modularization. In: *Proc. GECCO 2000*, pp. 1351–1358 (2000)
6. Seng, O., Bayrer, M., Biehl, M., Pache, G.: Search-based improvement of subsystem decomposition. In: *Proc. GECCO 2005*, pp. 1045–1051 (2005)
7. Seng, O., Stammel, J., Burkhart, D.: Search-based determination of refactorings for improving the class structure of object-oriented systems. In: *Proc. GECCO 2006*, pp. 1909–1916 (2006)
8. O’Keeffe, M., Ó Cinnéide, M.: Towards automated design improvements through combinatorial optimization. In: *Workshop on Directions in Software Engineering Environments – 26th ICSE*, pp. 75–82 (2004)
9. O’Keeffe, M., Ó Cinnéide, M.: Getting the most from search-based refactoring. In: *Proc. GECCO 2007*, pp. 1114–1120 (2007)
10. Harman, M., Tratt, L.: Pareto optimal search based refactoring at the design level. In: *Proc. GECCO 2007*, pp. 1106–1113 (2007)
11. Amoui, M., Mirarab, S., Ansari, S., Lucas, C.: A genetic algorithm approach to design evolution using design pattern transformation. *International Journal of Information Technology and Intelligent Computing* 1, 235–245 (2006)
12. Räihä, O.: *Genetic Synthesis of Software Architecture*, University of Tampere, Department of Computer Sciences, Licentiate thesis (2008)
13. Sahraoui, H.A., Godin, R., Miceli, T.: Can metrics help bridging the gap between the improvement of OO design quality and its automation? In: *Proc. ICSM 2000*, pp. 154–162 (2000)
14. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. *IEEE Transactions on Software Engineering* 20(6), 476–492 (1994)

Dual Phase Evolution and Self-organisation in Networks

Greg Paperin, David G. Green, and Tania G. Leishman

Faculty of Information Technology
Monash University, Clayton Campus, Building 63, Wellington Road
Clayton, 3800 Victoria, Australia
{Greg.Paperin, David.Green,
Tania.Bransden}@infotech.monash.edu.au

Abstract. Dual Phase Evolution (DPE) is a widespread natural process in which complex systems adapt and self-organise by switching alternately between two phases: a phase of global interactions and a phase of local interactions. We show that in evolving networks of agents, DPE can give rise to a wide variety of topologies. In particular, it can lead to the spontaneous emergence of stabilising modular structure.

Keywords: Dual Phase Evolution, networks, phase change, scale-free networks, modularity.

1 Introduction

Evolutionary computation research has mostly been concerned with the way individuals evolve. However, a growing challenge is to understand how networks evolve and adapt. From finance markets to the World Wide Web, from genetic regulatory networks to food webs, science and society are dominated by vast, complex networks of many kinds. Understanding how these networks acquire features and properties, such as stability and modularity, is crucial in trying to avoid catastrophic failures and other problems that emerge in such systems.

The problem of networks evolution raises many questions. For instance, what structures promote important properties such as resilience, stability, flexibility and modularity? What processes give rise to common topologies? And is it possible to direct network evolution to yield desired structures and system-wide behaviour?

Whilst combining local and global search is well known techniques in evolutionary computing, it is instructive to observe how they operate in natural systems. In this study we investigate a combination of global and local search by which complex networks are known to evolve *in nature*. In particular, we examine the role of Dual Phase Evolution (DPE) [1], a process wherein a network alternates between two distinct phases, in which different conditions prevail and different sorts of changes occur. Of particular interest is the potential of DPE to produce common network topologies found in natural systems, such as scale free networks, small-world networks, trees, and especially, modular networks.

Previous studies of network evolution have usually focused on the structures that emerge from the addition of nodes and edges in particular ways (e.g. [2, 3]). Here we investigate what kinds of network structures emerge when a given network is subjected to evolution in which phases of global interactions alternate with phases of local interactions.

2 Network Topologies

Networks (graphs) are sets of nodes and edges, with the edges linking pairs of nodes (e.g. computers on the internet). In recent years, research has revealed that processes forming networks often lead to particular network topologies. Some of these topologies have important properties and have been the subjects of intense research.

Random Networks

Not surprisingly, adding edges at random to an existing set of nodes leads to a random network. Erdős and Rényi [4] showed that when formed in this way, random networks exhibit a connectivity avalanche during which isolated nodes and small groups of nodes suddenly link to form a unique giant component. Essentially, they undergo a phase transition from a disconnected state to a connected state.

Trees and Scale-Free Networks

Networks that grow by attaching new nodes to existing nodes (by adding one edge only) form into trees. They have no cycles (Fig. 1.a). If in this process the new nodes attach preferentially to existing nodes with a large number of edges, then the result is a scale-free network [2]. The distribution of the number of edges connected to each node in the resulting network exhibits a power law, with most nodes having very small number of edges, and tending to group around a few nodes that have a large number of edges (Fig.1.b). As many observations of so-termed self-organised critical systems [5] also exhibit power law distributions, scale-free networks are often thought to underlie these systems (also see below).

Small-Worlds

These networks [3] form when long distance connections are added at random to regular networks (Fig.1.c). They are characterized by low paths lengths between nodes and high clustering. Some social networks take this form.

Modular Networks

Modular structures are the most common way by which natural and artificial systems reduce complexity. An unresolved question is what processes form modular networks? A modular network is one in which sets of nodes form highly connected groups (modules) that are only weakly connected to the rest of the network (Fig. 1.d).

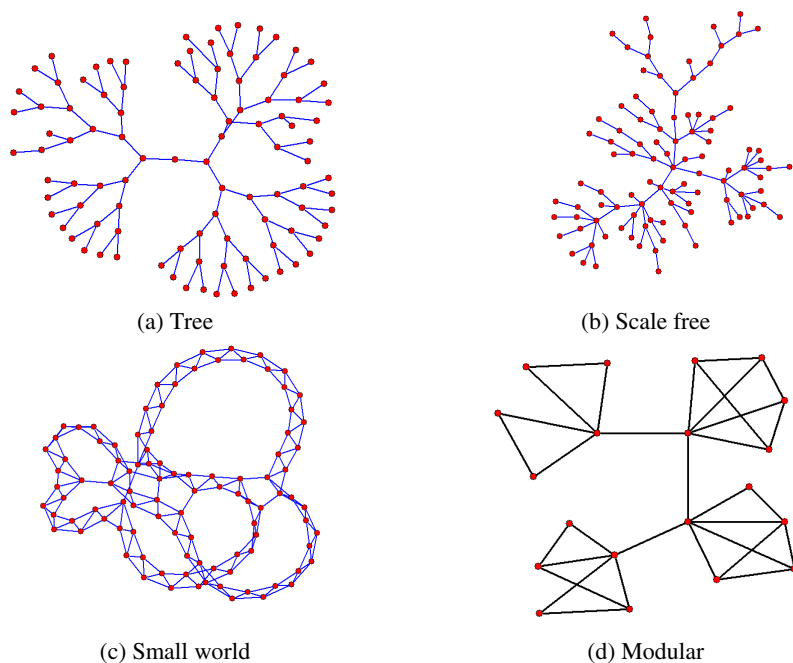


Fig. 1. Some common structures in networks. **(a)** A tree has branches, usually from a root node, and no cycles (loops). **(b)** A scale free network has a negative exponential distribution of edges per node. **(c)** A small world has a regular structure of local connections with some randomly placed long-range connections. **(d)** A modular network consists of component networks (modules) that are well connected internally, but poorly connected to the rest of the network. The example shown here has four modules.

3 Dual Phase Evolution (DPE)

An important consequence of the connectivity avalanche is that networks can exist in two phases: “connected” and “disconnected”. The “connected” phase is signified by high edge density, and short paths lengths. In this phase interactions can therefore occur between most of the network components. In the “disconnected” phase the edge density is low, the paths lengths are high, and the network typically consists of several disconnected components. Interactions in the disconnected phase typically occur locally or only within strongly connected components.

Since networks are inherent in the structure and behaviour of all complex systems [6], the connectivity avalanche underlies many kinds of critical phase changes [7]. Therefore all such systems can switch between the two above phases. Systems in the disconnected phase tend to be balanced. They exhibit little large-scale variations and intrinsic perturbations cannot propagate far. Systems in the connected phase, in contrast, tend to exhibit significant variation on all scales. Their rich connectivity allows perturbations to propagate far and to affect many parts of a system [1].

Dual Phase Evolution (DPE) occurs when an evolving system repeatedly switches between these two phases. Different processes operate in each phase. In previous studies [1, 8-10], we have found that many systems affected by DPE exhibit the following properties (see also Fig. 2):

1. Systems in a balance phase (phase of predominantly local interactions) are stabilised through the interactions of local system components on various levels of control. In biological evolutionary systems selection is a strong factor that stabilises systems during the balance phase.
2. For systems in a variation phase (phase of global interactions) natural selection may play a very different role by facilitating and enhancing the random variation of local components, for which chaos acts as a source of novelty. Rich connectivity allows intrinsic and extrinsic impulses to propagate far and to affect systems on various scales.
3. Phase changes are often mediated by perturbations or external stimuli.
4. Complexity accumulates as a result of repeated phase changes. During a variation phase, new configurations and interactions are discovered for stable system modules. While the connectivity consequently falls, unstable interactions and poorly-adapted designs are selected out, allowing increasingly complex, stable and orderly structures to crystallise into a new balance phase.

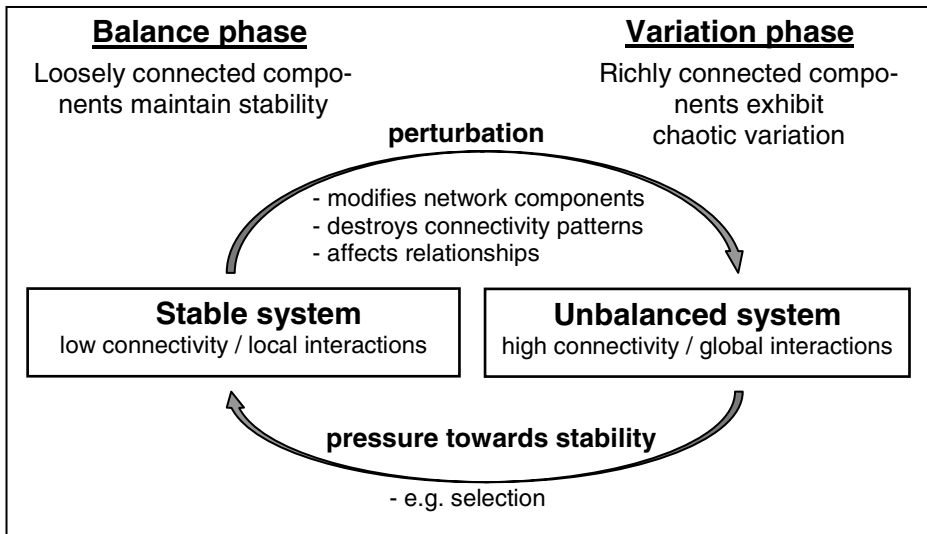


Fig. 2. The mechanism of Dual Phase Evolution. Systems flip between loosely connected balance and well connected variation phases. Perturbations and external stimuli unbalance stable systems, variation facilitates evolutionary exploration, and internal pressures drive the system into a new stable state.

Contrast between DPE and Self-Organised Criticality

In contrast to DPE, the notion of Self-Organised Criticality (SOC) [5] suggests that many networks underlying natural phenomena self-organise to a critical state in which

the connectivity remains within the critical transition region between the connected and the disconnected states. In this critical state, small changes in the local network structure may have large-scale effects on the global structure, and the “size” of such effects often follows a power-law distribution. The extent to which SOC presents a general form for organisation of complex networks remains unclear. As shown in this paper, DPE processes can also lead to the emergence of scale-free networks. In fact, in many cases there are several processes which may lead to power-law distributed data. At this stage, we merely aim to pinpoint the key difference between DPE and SOC:

The SOC view is that systems self-organise towards a critical “edge of chaos” region. To describe DPE using the SOC-vocabulary, – systems develop to a balance-state, in which the underlying networks exhibit low density and high stability. External disturbances and other forces repeatedly push a system across the critical region into a chaotic state (in the sense that the system’s responses to random stimuli and variations are unpredictable), in which the underlying networks are densely connected and unstable. From here the system returns to a new balance-state, accumulating order and complexity on the way [1].

4 DPE and Network Topology

DPE has the capacity to form any of the network topologies described earlier. The two phases involved make it possible to combine different mechanisms in a rich variety of ways. We conducted a series of experiments in order to demonstrate how particular network topologies can arise through DPE processes.

Scale-Free Networks

It has been shown [2] that in growing networks scale-free structures can arise when new nodes are preferentially connected to existing nodes of higher degree, however, it has not been previously shown how such structures can emerge in networks of constant size. Here we show that a scale-free network can form from a random network through a DPE process in which edges are added between well-connected nodes and edges are removed between poorly connected nodes.

To demonstrate this we created a computer simulation consisting of a network of N nodes. We initialised the network with random connections. Sensitivity analysis has shown that various initial edge densities between 1% and 50% do not affect the following results; however, a large initial edge density can increase the time to convergence. Thereafter we have subjected the network to DPE-structural modifications:

Local phase: During this phase nodes were interacting with other directly neighbouring (i.e. connected) nodes. First, a random node a was selected with a probability proportional to the number of edges connected to a :

$$p_{L,s}(a) = \frac{\text{degree}(a) + 1}{1 + \sum_{i=1}^N (\text{degree}(n_i) + 1)} \quad (1)$$

where $p_{L,s}(a)$ is the probability of the node a being selected and $\text{degree}(n_i)$ is the number of edges connected to a node n_i . Then, if a had any connected neighbours, a neighbour b of a was randomly selected with a probability anti-proportional to the number of edges connected to b :

$$p_{L,e}(b) = 1 - \frac{\text{degree}(b) + 1}{1 + \sum_{i=1}^d (\text{degree}(k_i) + 1)} \quad (2)$$

where $k_1..k_d$ are the nodes connected to a . Then, the edge (a, b) was removed. In this model, high-degree nodes preferentially interact with their low-degree neighbours.

Global phase: During this phase nodes could interact with any other nodes within the network. First, a node c was randomly selected with a probability anti-proportional to the number of connected edges:

$$p_{G,s}(c) = 1 - \frac{\text{degree}(c) + 1}{1 + \sum_{i=1}^N (\text{degree}(n_i) + 1)} \quad (3)$$

Then a second node d was chosen randomly and independently of c with a probability proportional to the number of edges connected to d :

$$p_{G,e}(d) = \frac{\text{degree}(d) + 1}{1 + \sum_{i=1}^N (\text{degree}(n_i) + 1)} \quad (4)$$

If there was no edge (c, d) then it was added to the network, otherwise a new pair of nodes was selected. In this model low-degree nodes preferentially interact with high-degree nodes.

We have run this model for several thousand iterations. A phase change occurred every 10 iterations, however, sensitivity analysis has shown that the results remain unaffected as long as the phases are approximately equal in duration and as long as a phase change occurs before the edge density gets extremely high or falls to zero. At every iteration, we measured the edge density D and the goodness of fit R^2 of the degrees of the network nodes to a power distribution.

The results (Fig. 3) clearly show that the edge density of a network subjected to the above process converges on a certain value D_s that depends on N . E.g., for $N = 100$, $D_s \approx 0.06$; for $N = 200$, $D_s \approx 0.04$; for $N = 1000$, $D_s \approx 0.003$. In the initial random network, the node degrees are uniformly distributed and R^2 is close to zero. As D approaches D_s , R^2 experiences a critical transition to a value close to 1. A plot of a typical run can be seen in Fig. 3. The relationship between N and D_s , and the critical transition of the network from random to scale-free when subjected to DPE is the subject of ongoing investigations. Here we concentrate on how further network structures can arise under DPE-conditions.

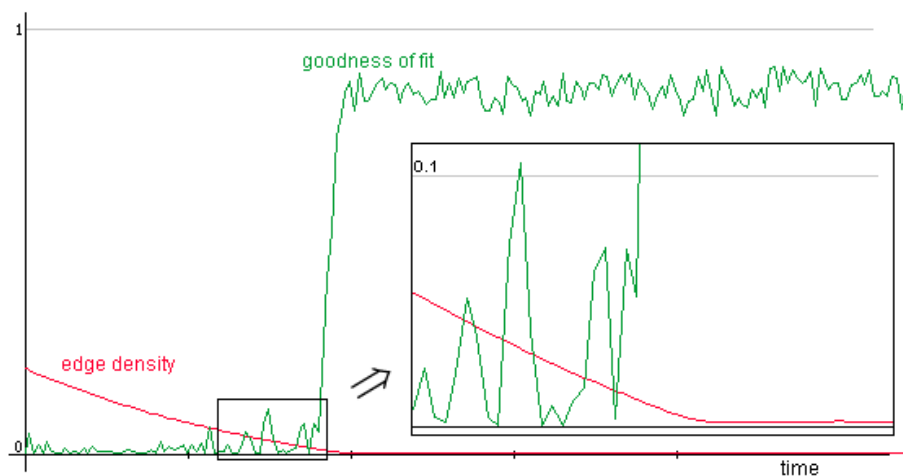


Fig. 3. Critical transition of the node degree distribution from random to scale-free. Depicted is a typical run for a network of $N=1000$ nodes subjected to a DPE-process. Edge density D stabilises around ca. 0.0003. (Image created using the LiveGraph data analysis system [11]).

Circuits and Trees

Tree and circuit structures can also evolve in a network of constant size that is subjected to DPE. To demonstrate this we created a computer model of a network in which nodes interact globally and locally building new links that decay over time. In our model each node has a number of binary attributes and each edge has an associated weight. When two nodes interact, they establish a link (edge) if they are not connected, or reinforce the link by an amount w_i if they are. At each iteration, the strengths of all links decay by an amount w_d (w_d is by an order of magnitude smaller than w_i). If the strength of a link falls to zero, it is removed.

Local phase: During this phase the nodes interact with other nodes to which they are connected or with the nodes connected to their neighbours. In addition, nodes also interact with other unlinked nodes, if the interaction partners match in the values of most of their binary attributes.

Global phase: During this phase all nodes interact randomly. In this model the duration of the global phase is by an order of magnitude shorter than the duration of the local phase.

Two typical resulting networks are depicted in Fig. 4. There is an intriguing parallel between this experiment and the mechanisms involved in the emergence of social networks. People normally interact with other people with whom they are already acquainted or with likeminded people and they strengthen the mutual relationships through these interactions. On rare occasions, such as social or business meetings or conventions, people interact with anyone who they encounter during the event thus forming new links. When a personal relationship is not maintained through repeated contact it slowly decays and eventually disappears. Recently, a research paper [12] attracted much attention, in which the romantic relationships between students at a US

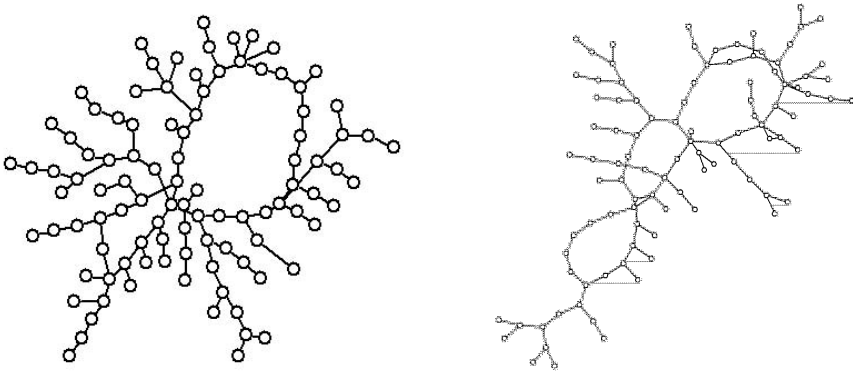


Fig. 4. When nodes have multiple attributes, DPE gives rise to networks, consisting chiefly of chains and branches (and some large cycles), similar to those observed in human societies

high school that occurred during a 6-months period were mapped and represented as a network. The similarity between that network and Fig. 4 is intriguing.

Modular Networks

Of greatest interest in this study is the formation of modular networks. Modular structures play a central role in dealing with complexity in most kinds of systems. In modular networks nodes are encapsulated within small, local sub-network. This prevents a wider range of interactions and thus inhibits the emergence of unexpected, and often disastrous, effects. The local modules are often richly connected and stable, while the inter-modular connections are often less enduring. This is a crucial property that allows network modules to be used in larger and more complex networks and systems.

The mechanisms behind the formation of modular networks in nature are not well understood. Here we suggest that DPE may be the general mechanism for the emergence of such structures. To demonstrate this we modified the above model used for the formation of circular networks. In the new model each node has only one attribute, but that attribute may assume one of many discrete values. In addition, the edge weight modifiers were set to values that differed only by a small amount. The sensitivity of the model to this setting is the subject of on-going investigations.

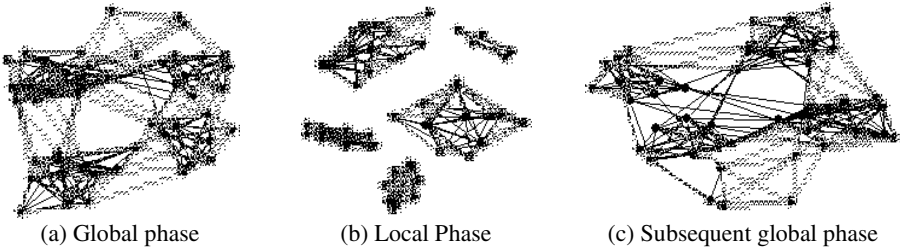


Fig. 5. Phases changes in a modular network over time. See text for explanation.

This DPE process quickly leads to formation of modules in an initially random network. The continuous reinforcement of local links and the decay of intra-modular links typical for balance phases maintain a stable overall structure and may cause the modules to become disconnected from each other over time. During the variation phases the modules become connected in new ways forming novel and complex structures (Fig. 5).

5 Conclusion

From genetic regulation to financial markets to global communications, many of the great challenges facing us today involve networks of great complexity. Traditional methods of pre-designing systems often prove inadequate when faced with such complexity. Unexpected and sometimes disastrous features emerge out of the myriads of interactions between the nodes that make up real network. Stock market crashes and cascading failures in power grids are just two examples.

In contrast, many living systems evolve network patterns that achieve the desired stabilising effects. As we described earlier, research in recent years has identified a number of commonly found network patterns that emerge as networks grow and evolve. This raises the possibility of evolving, rather than designing complex networks. That is, allow systems to adapt and change over time so that desired properties emerge out of the interactions between the nodes.

In this brief study we have shown that under appropriate conditions, a single general mechanism, Dual Phase Evolution, is capable of generating a wide variety of network structures, including all of the common structures that have been investigated in recent years. Some of these, such as scale-free networks, have already been intensively studied. Here we have shown that scale-free networks can not only emerge as they first grow and form, but also from an existing network that adapts and evolves.

A further important finding is that modular structures can form under Dual Phase Evolution. Modularity is the most widespread mechanism by which natural and artificial systems deal with complexity. In evolutionary computing, modularity can play an important role in optimisation and other algorithms. Modular genetic algorithms, for instance, solve complex problems by first solving simpler problems that use smaller sets of genes. Likewise, modular neural networks include network modules, smaller networks that have been trained to solve simpler problems.

At present, however, modular methods in evolutionary computing make use of pre-existing modules, which requires prior knowledge of the nature of the problem to be solved. In contrast, our results suggest that in algorithms employing a DPE approach, appropriate modules may emerge in the course of solving a complex problem.

Acknowledgment

This work was funded in part by the Australian Research Council.

References

1. Paperin, G., Green, D.G., Sadedin, S., Leishman, T.G.: A Dual Phase Evolution model of adaptive radiation in landscapes. In: Randall, M., Abbass, H.A., Wiles, J. (eds.) ACAL 2007. LNCS (LNAI), vol. 4828, pp. 131–143. Springer, Heidelberg (2007)
2. Albert, R., Barabási, A.L.: Topology of Evolving Networks: Local Events and Universality. *Physical Review Letters* 85, 5234–5237 (2000)
3. Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world' networks. *Nature* 393, 409–410 (1998)
4. Erdős, P., Rényi, A.: On the Evolution of Random Graphs Magyar Tudományos Akadémia. *Matematikai Kutató Intézetének Közleményei* 5, 17–61 (1960)
5. Bak, P.: *How Nature Works: The Science of Self-Organized Criticality*. Springer, Heidelberg (1999)
6. Green, D.G.: Emergent Behaviour in Biological Systems. In: Green, D.G., Bossomaier, T.R.J. (eds.) *Complex Systems: From Biology to Computation*, pp. 24–33. IOS Press, Amsterdam (1993)
7. Green, D.G.: Self-Organization in complex systems. In: Bossomaier, T.R.J., Green, D.G. (eds.) *Complex Systems*, pp. 7–41. Cambridge University Press, Cambridge (2000)
8. Green, D.G., Kirley, M.G.: Adaptation, diversity and spatial patterns. *International Journal of Knowledge-Based Intelligent Engineering Systems* 4, 184–190 (2000)
9. Green, D.G., Leishman, T.G., Sadedin, S.: Dual phase evolution: a mechanism for self-organization in complex systems. *InterJournal Complex Systems* (2006)
10. Kirley, M.G., Li, X., Green, D.G.: The cellular genetic algorithm. In: McKay, B., Yao, X., Newton, C.S., Kim, J.-H., Furuhashi, T. (eds.) SEAL 1998. LNCS, vol. 1585, pp. 93–100. Springer, Heidelberg (1999)
11. LiveGraph - a framework for real-time data visualisation, analysis and logging (accessed on 01.03.2008), <http://www.live-graph.org>
12. Bearman, P.S., Moody, J., Stovel, K.: Chains of Affection: The Structure of Adolescent Romantic and Sexual Networks. *American Journal of Sociology* 110, 44–91 (2004)

Heterogeneous Payoffs and Social Diversity in the Spatial Prisoner's Dilemma game

Golriz Rezaei and Michael Kirley

Department of Computer Science and Software Engineering,
The University of Melbourne,
Victoria, Australia

`{grezaei,mkirley}@csse.unimelb.edu.au`

Abstract. In this paper, we investigate the role of heterogeneous payoff values and social diversity in a spatial version of the iterated prisoner's dilemma game. Typically, a fixed number of agents play the game over a specified number of rounds. At each time step, the agents receive a fixed reward based on the strategy they have adopted and the corresponding payoff (or reward) matrix. We argue that such restrictions are unlikely to be fulfilled in real-life situations. Subsequently, we introduce additional features into the game. Here, each agent has an additional age attribute that can be used to control the number of iterations of the game an agent actually participates in. We also introduce dynamic payoff values that are correlated with particular agent experience levels. Numerical simulations show that the proposed heterogeneous agent model promotes the evolution of cooperation in some circumstances.

1 Introduction

A game-theoretic approach for the study of evolution of cooperation is typically based on the Prisoner's Dilemma game [1,13]. The game consists of two players (agents), each of which can adopt one of two strategies: to cooperate C or to defect D . If both players cooperate they are rewarded. Alternatively, if they both defect they are punished. If one player defects while the other cooperates, they get the “temptation” or “sucker's” payoff respectively. Relative payoffs are shown in Table 1.

The inequality $T > R > P > S$ constrains the game, such that defection is worth more than cooperation, and mutual cooperation is better than mutual defection. The dilemma arises in the game as a consequence of the self-interested behaviour of the agents. Independently of what the other player does, defection, D yields a higher payoff than cooperation C and is the dominant strategy. The outcome (D,D) is therefore a Nash equilibrium – despite the knowledge and awareness of the dilemma, both players opt to defect even though both know they are going to receive inferior outcomes. In addition, the inequality $2R > (T + S)$ is usually employed so that players are not collectively better off if they simply alternate between playing C and D .

Table 1. Prisoner’s Dilemma Game Payoff Matrix

	C	D
C	(R,R)	(S,T)
D	(T,S)	(P,P)

An interesting variation of the game is the Iterated Prisoner’s Dilemma, where the agents play repeatedly [1,2]. Iterative games represent a more realistic approximation of the real-world, as truly “one-shot” games are comparatively scarce. However, if we limit the iterations to a fixed, predetermined number of rounds then defection may be a dominant strategy [22]. Consequently, to evolve cooperation it is necessary to run the game infinitely, or the length of interactions should be sufficiently large to encourage cooperation.

In the spatial version of the Iterated Prisoner’s Dilemma game, the agents are usually mapped onto a regular lattice. Interactions between agents are restricted to the local neighborhood [5,8,11,12,17]. The utility (fitness) of each individual is determined by summing its payoffs in games against each of its neighbors. The scores in the neighborhood, including the individual’s own score are typically ranked. In the following round (iteration) the individual adopts the strategy of the most successful agent from the local neighborhood. Importantly, the spatial structure allows cooperators to build clusters in which the benefits of mutual cooperation can outweigh losses against defectors [11]. From a biological perspective, the utility of an individual is interpreted in terms of reproductive success. Alternatively, from an economic perspective the utility refers to individuals adapting their strategy to mimic a successful neighbor.

It is well documented that the level of emergent cooperation in the spatial versions of the game is sensitive to many parameters, including the magnitude of the rewards/punishments, population size, initial condition and update rules [2,5,8,11]. Typically, however, the magnitude of the rewards/punishments for playing a particular strategy are fixed in a given game (depending upon the strategy played by the opposing agent) and the same fixed number of agents are active throughout the game period. We argue that such restrictions are unlikely to be fulfilled in real-life situations where diversity is ubiquitous [16] . If varying levels of “social diversity” are introduced into the game via dynamic payoff values [3,19,14] or via stochastic updates [12,20] the trajectory of the population will be altered. This in turn will impact on the level of cooperation found in the population.

In this study, we extend the standard spatial Iterated Prisoner’s Dilemma game in a number of interesting ways. In our model, each agent has an additional age attribute and life span. Although the total number of agents playing the game is kept constant, agents may die and can be replaced by new agents. We also introduce dynamic payoff values into the game. Here, each agent has their own version of the payoff matrix realized by introducing scaling factors that determine the mapping from game payoffs to the fitness of individual players [14]. The rationale for this approach is based on the fact that associated payoffs

from pursuing certain strategies vary based on an agent's individual experience. For example, in biology factors such as size, age, health, expected life span (eg. [21]), available mates, behaviour (personality traits) may affect the cost and/or benefit of playing a particular strategy. Similarly, in economics personal preference, socio-economic status and past experience may influence decisions made and the corresponding rewards [16].

The remainder of the paper is organized as follows: In the next section, we briefly review related work using dynamic payoff matrices in alternative versions of the Prisoner's Dilemma game. In Section 3, we describe our proposed model using dynamic payoff values and social diversity based on agent experiences. In Section 4, we present detailed numerical simulation results. The paper concludes with a summary of the implications of this work.

2 Related Work: Dynamic Payoff Values

There have been relatively few studies that have examined the impact of varying the magnitude of the payoff matrix values in the Prisoner's Dilemma game. Three recent examples from the literature based on the spatial game are briefly discussed below:

Tomochi and Kono [19] introduced an interesting model in which the payoff matrix was designed to evolve depending on the ratio of defectors (or cooperators) to the whole population. However, the dynamic mechanism, which altered the values of P and R with respect to time, produced a universal payoff matrix applicable to all members of the population at time t . In this model, the inequalities defining the Prisoner's Dilemma were always maintained. The level of cooperation within the population was directly related to the payoff matrix values.

Perc and Szolnoki [14] have investigated the effects of social diversity in the spatial prisoner's dilemma game. In that study, random noise drawn from alternative statistical distributions was added to the payoff matrix at the beginning of the game. They concluded that this correlated "social diversity mechanism" promoted higher-levels of cooperation in the spatial game examined. It was suggested that variable social status might play a crucial role in the evolution of cooperation.

Fort [3] introduced a model in which the payoff matrix was correlated with a spatial and temporal zones. The heterogeneous environment was constructed by varying the value of T based on both geographical location and simulated time. Here, it was possible that the payoffs for an agent and their opponent were not equal – reminiscent of what happens in general in real life. The results reported suggested that the effect of asymmetries in the interactions between agents, which takes into account the effect of asymmetries in the costs and benefits on the evolution of cooperation, had a direct impact on the proportion of agents cooperating in the population. As expected, the model was sensitive to the value of T used. In Fort's study, the Prisoner's Dilemma inequality was relaxed, and when the payoff matrix values changed, the game oscillated between

the Prisoner's Dilemma game and Chicken game ($T > R > S > P$) or the game becomes Stag Hunt game ($R > T > P > S$).

It is clear from the studies report above that the relative payoff rankings at particular instances of time impact on the emergent properties of the game. However, there are still many open questions related to what impact (if any) dynamic payoff values have especially when each agent maintains their own version of the payoff matrix when situated in a constrained game environment.

3 The Model

A key feature of the extended spatial game used in this study was the introduction of a social diversity framework based on the age of a particular agent and dynamic reward system (or rescaled payoff values). We start by introducing the **homogeneous model** with fixed payoff values. This is followed a description of the **heterogeneous model** with dynamic payoff values.

3.1 Homogeneous Model

The underlying framework was similar to the standard spatial Iterated Prisoner's Dilemma: the agents (players) were mapped onto the nodes of a 2-D regular lattice (size $n \times n$) with periodic boundary conditions. At each iteration, an agent played a game with each of its local neighbors. Here, we have used the eight-node Moore local neighbourhood. This model assumes that an individual located at node n , playing the row strategy r at time t receives the payoff $p_{r,c}$ if it interacts with a neighbour m playing the column strategy c (See Table 1 for the payoff values).

To implement the enhancements proposed in our model, it was necessary to introduce additional attributes, age α_i and life span λ_i , for each agent. Each agent was randomly allocated a life span λ_i drawn from a uniform distribution (see parameter values in the next section). This meant that each agent i , participated in the game from its birth (introduction into the game) until it reached its expected life span λ_i , when it dies. At that point, it was replaced by a new randomly generated agent k .

An agent maintained a cumulative payoff throughout its life span – its *utility*. However, to cater for the age differences between agents in the population, we have used the normalized utility μ_i for selection purposes. Here, μ_i was simply the utility of the agent divided by its age. This particular use of normalized payoffs is similar to the so-called participation costs [15]. Obviously, older agents have had more opportunity to interact and accumulate higher utility values. Consequently, by using the normalized utility for selection, younger agents have an opportunity to be competitive.

After each iteration of the model, poorly performing agents (low μ_i) were presented with an opportunity to update their strategy deterministically. The fitness value of each individual was compared to all of its neighbors. The agent then updated its strategy by imitating the strategy of their eldest neighbour with

the highest μ_i value. In case of a tie between two agent neighbours, a random agent was chosen to imitate. All agents in the model have an opportunity to update their strategy asynchronously once in each iteration.

3.2 Heterogeneous Model

In the heterogeneous model, we introduced dynamic payoff values into the game. Here, each agent maintained its own payoff matrix that changed over time.

At each iteration of the model (round of the game) rescaled payoff values, which constitute the payoff matrices of the game, were calculated in a similar fashion to the methods proposed by Perc and Szolnoki [14]. The actual equations are described below:

$$X_{i,t} = X(1 + \xi_{i,t}) \quad (1)$$

$$\xi_{i,t} = \kappa \times \left(\frac{1}{1 + \exp[\alpha_{i,t}/\lambda_i]} \right) \quad (2)$$

where: $X_{i,t}$ is the payoff value for agent i at time t ; X is the default payoff matrix values T, R, P or S ; $\xi_{i,t}$ is the magnitude of the rescaled values; $\alpha_{i,t}$ is the age of agent i at time t ; λ_i is the expected lifetime of agent i , and κ characterizes the uncertainty related to the environment (or social diversity), it is used as a limiting factor based on agent experience.

From equations 1 and 2, it can be seen that the magnitude of the rescaled payoff values is dependent on the value of $\xi_{i,t}$. Therefore, at time t it is possible that all agents have a different matrix (although, still a valid Prisoner's Dilemma game). If we consider Table 1 as the base case, agent i would receive the corresponding row payoff values and agent j would receive the column payoff from their individual matrix.

4 Simulation Experiments

In this study, a number of fundamental questions related to the evolution of cooperation in heterogeneous agent population have been considered, including: (a) what impact do the dynamic payoff values have on the level of cooperation in the population? (b) how does the age-based social diversity parameter impact on the evolutionary trajectory? (c) what type of long-term behaviour arises, and how long does it take to arise? The underlying hypothesis tested is that the heterogeneous social state of the agents promotes the evolution of cooperation.

Our model was implemented using Netlogo4.0 [10]. The analysis of the model was based on systematic Monte Carlo simulations, performed using dynamic payoff matrix values described above. In all simulations, the number of nodes in a regular lattice was fixed at 32×32 . We report results when the population was initialised with 20% cooperators and 80% defectors. (We have tested a range of alternative population sizes and initial strategy proportions with no significant difference recorded between them). Each trial was run for 1000 iterations. All configurations were repeated 30 times with statistical results reported below.

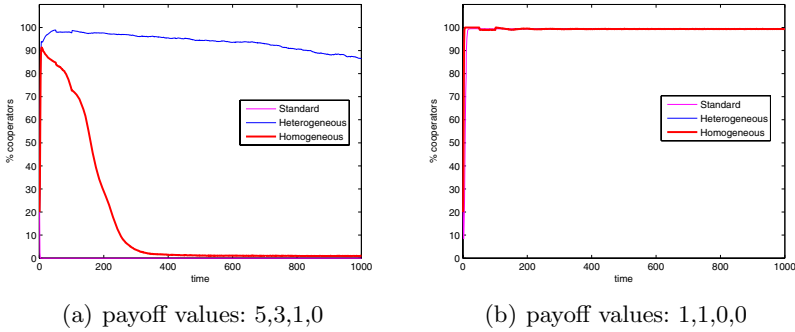


Fig. 1. Proportion of cooperators vs time for two different base-level payoff values. (a) $T = 5$, $R = 3$, $P = 1$, $S = 0$ (b) $T = 1$, $R = 1$, $P = 0$, $S = 0$. Error bars have been omitted for clarity as the values are extremely small.

In the first set of experiments, we have examined the sensitivity of our model using a range of different base payoff values. Fig. 1 plots results for two different base level payoff values: (a) $T = 5$, $R = 3$, $P = 1$, $S = 0$ and (b) $T = 1$, $R = 1$, $P = 0$, $S = 0$ for three different implementations: the standard Iterated Prisoner's Dilemma game, our model limited to homogenous payoff values (normalized utility values are used, all agents use the same payoff matrix, and λ was drawn from a uniform distribution [50-100]) and the full heterogeneous model, where $\kappa = 0.2$. An examination of the plots shows that the social diversity model typically leads to higher levels of cooperation for relatively large base payoff values (see fig. 1a). In the standard game, the whole population converged to defection (fig. 1a) and cooperation (fig. 1b) within twenty time steps (this is difficult to see on the plot as the values almost lie on the axes). The homogeneous model is typically able to foster higher levels of cooperation for longer when compared to the standard model. The resulting level of cooperation in the heterogeneous model is significantly greater (p-value < 0.05) than the other models. For relative small payoff values (where T and R are approximately equal), there was no significant difference between the results.

In the next set of experiments, the magnitude of κ was systematically changed. κ is used as a limiting scale factor for $\xi_{i,t}$ and thus represents an environmental constraint on social diversity inherent in the model. Smaller values of κ result in a smaller range of values in each agent's payoff matrix. In fig 2, five different time series are displayed when the base payoff values were $T = 5$, $R = 3$, $P = 1$, $S = 0$ and λ was drawn from a uniform distribution [50-100]. The resulting time series values are in the same order as the κ values. For larger κ values, the proportion of cooperators in the population is significantly greater than for lower values.

Fig. 3 provides a series of snap-shots of the evolving population for the homogeneous and heterogeneous models respectively when $\kappa = 0.1$. Here, four different time values are displayed. In both models, the initial proportion of cooperators was 20%. At $t = 200$ differences between the proportions of strategies

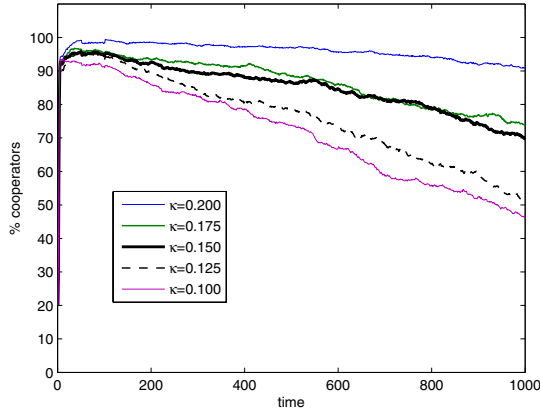


Fig. 2. The impact of κ on the resulting levels of cooperation. Error bars have been omitted for clarity as the values are extremely small.

are apparent. These differences are magnified further at $t = 600$. By the end of the trial ($t = 1000$), the population in the homogenous model consists of almost all defectors. In contrast, the number of cooperators in the heterogeneous model is significantly higher. An inspection of the plot reveals clusters (of varying size) of cooperators.

The results described above, clearly show that the resulting levels of cooperation are dependent on both the magnitude of the rescaled payoff and base payoff values used. The social diversity clearly alters the trajectory of the population. In the final experiment, we establish how alternative life span distributions affect the final results in the heterogeneous model. Here, we have examined three different scenarios where the value of λ for each agent was randomly drawn from different uniform distributions: $[0-50]$, $[50-100]$ and $[0-100]$.

Fig. 4a plots the corresponding time series values when $\kappa = 0.2$ and the base payoff values were $T = 5$, $R = 3$, $P = 1$, $S = 0$. As expected, there are significant differences between the scenarios. Agents die when they reach their λ value and are replaced by randomly generated agents. If the value of λ is relatively small $[0-50]$, the frequent changes in the population members limit the positive effects of the dynamic rescaled payoff values. In contrast, when agents are provided with an opportunity to “live” for longer (a wider range of λ values $[0-100]$), the resulting proportion of cooperators tends to be higher. If the range of λ values is biased towards larger values $[50-100]$, higher proportions of cooperators in the final population are evident. However, this may be attributed to the fact that a relative large number of agents playing a randomly generated strategy have been added to the population. In fig 4b, we compare scenarios when the agent at the end of their life span was replaced either randomly or by a defector in the both models ($\kappa = 0.2$). The model is sensitive to the update rule followed. Clearly, the heterogeneous model promotes higher levels of cooperation even when the agents are replaced by other agents playing only the defect strategy.

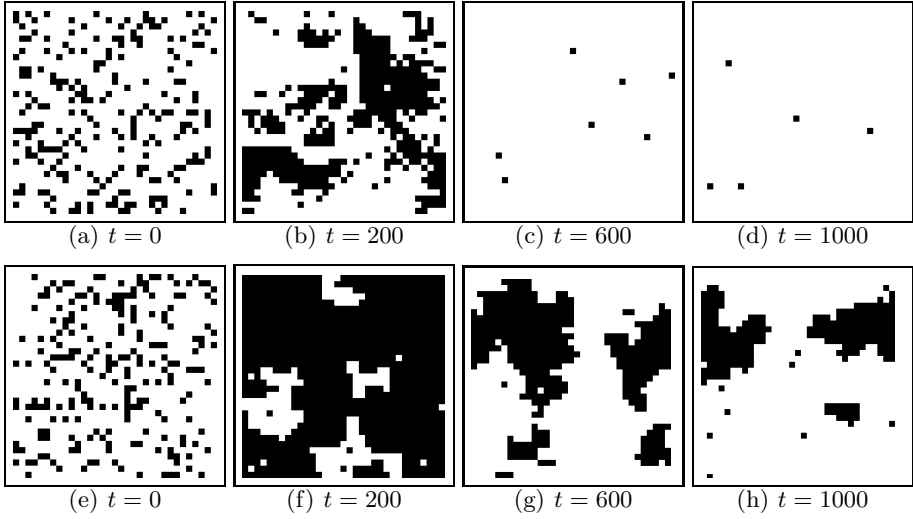


Fig. 3. Snap-shots of the evolving population in a typical simulation trial. The top row plots the homogeneous model. The bottom row plots the heterogeneous model. Four different time steps are recorded for each model. The white cells represent the defectors, black cells represent the cooperators.

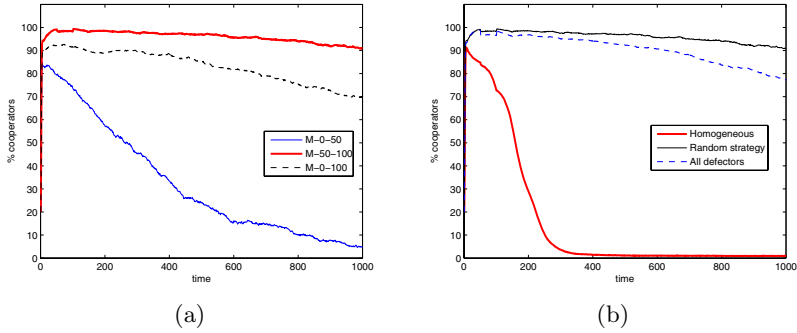


Fig. 4. Proportion of cooperators vs time for different scenarios. (a) The impact of the λ distribution on the resulting levels of cooperation in the heterogeneous model. (b) The impact of different replacement rules. Error bars have been omitted for clarity as the values are extremely small.

5 Discussion and Conclusion

In this paper, we have investigated the population dynamics of a version of the spatial Iterated Prisoner's Dilemma game with heterogeneous payoffs and social diversity. An important difference between this study and previous work was the fact that each agent was equipped with their own payoff matrix, which was

realized by introducing scaling factors on the base payoff values. The rescaled values were a function of the agent's age or experience level. The rationale for this approach was based on the fact that "socio-economic" status and past experience may influence decisions made and the corresponding rewards. In this study, we have also relaxed the constraint that the same agents participate in each iteration of the game. In our model, when an agent reached its expected life span, it would die and be replaced by a randomly generated agent.

The specific aim of this study was to examine the variation in the proportion of particular strategies in the agent population playing at equilibrium levels, based on dynamic payoff values. To meet the specific objectives, systematic numerical simulations were used to determine how alternative parameter settings influenced the system dynamics.

In all simulation trials, the average frequencies of cooperators approach equilibrium values irrespective of the initial conditions. A detailed sensitivity analysis revealed that the proportion of cooperating agents in the system was sensitive to the magnitude of the rescaling value $\xi_{i,t}$ (and κ in particular) in both the homogeneous and heterogeneous models. Typically, the proportion of cooperators was higher in the final population when the dynamic payoff values were used.

We have argued that it is unrealistic to restrict the number of agents playing the game to a fixed number of agents with fixed payoff values. The simulation results suggest that variations in the magnitude of payoff values is highly dependent on the system state, and in our case, the age and/or experience level of the agents. Some agents may incur relatively lower costs in contributing to the "public good" which, in turn may lead to a better overall outcome for the system even when some defectors persist. Alternatively, some agent may not incur a high cost for defecting. Our results provide further supporting evidence that variable "social status" plays an important role in the evolution of cooperation in multi-agent systems.

In future work, we will extend this model and investigate the application of this abstract dynamic payoff framework to distributed multi-agent systems applications: specific domains to be investigated include peer-to-peer network applications (file sharing) and mobile adhoc networks.

References

1. Axelrod, R.: The Evolution of Cooperation. Basic Books, New York (1984)
2. Axelrod, R.: Evolution of cooperation. *J. Conflict Resolution* 24, 3–25 (1980)
3. Fort, H.: On evolutionary spatial heterogeneous games. *Physica A* (2007)
4. Grim, P.: The greater generosity of the spatialized prisoner's dilemma. *J. theor. Biol.* 173, 353–359 (1995)
5. Hauert, C.: Fundamental clusters in spatial 2×2 games. *Proc. R. Soc. Lond. B* 268, 761–769 (2001)
6. Hauert, C., Doebeli, M.: Spatial structure often inhibits the evolution of cooperation in the snowdrift game. *Nature* 428, 643–646 (2004)
7. Ifti, M., Killingback, T., Doebeli, M.: Effects of neighborhood size and connectivity on the spatial Continuous Prisoners Dilemma. *Journal of Theoretical Biology* 23, 97–106 (2004)

8. Lindgren, K., Nordahl, M.G.: Evolutionary dynamics of spatial games. *Physica D* 75, 292–309 (1994)
9. Masuda, N., Aihara, K.: Spatial prisoner's dilemma optimally played in small-world networks. *Physics Letters A* 313, 55–61 (2003)
10. Wilensky, U.: NetLogo is a cross-platform multi-agent programmable modeling environment. In: *Modeling Nature's Emergent Patterns with Multi-agent Languages. Proceedings of EuroLogo 2002* (2002), <http://ccl.northwestern.edu/netlogo/>
11. Nowak, M.A., May, R.M.: Evolutionary games and spatial chaos. *Nature* 359, 826–829 (1992)
12. Nowak, M.A., May, R.M.: The spatial dilemmas of evolution. *Int. J. Bifurcation Chaos* 3, 35–78 (1993)
13. Nowak, M.A., Sigmund, K.: Tit for tat in heterogeneous populations. *Nature* 355, 250–252 (1992)
14. Perc, M., Szolnoki, A.: Social diversity and promotion of cooperation in the spatial prisoner's dilemma game. *Physical Review E* 77, 011904 (2008)
15. Santos, F.C., Pacheco, J.M.: A new route to the evolution of cooperation. *Journal of Evolutionary Biology* 19(3), 726–733 (2006)
16. Santos, F.C., Santos, M.D., Pacheco, J.M.: Social diversity promotes the emergence of cooperation in public goods games. *Nature* 454, 213–216 (2008)
17. Schweitzer, F., Behera, L., Muhlenbein, H.: Evolution of Cooperation in a Spatial Prisoner's Dilemma. *Advances in Complex Systems* 5(2-3), 269–299 (2002)
18. Tomochi, M.: Defectors niches: prisoner dilemma game on disordered networks. *Social Networks* 26, 309–321 (2004)
19. Tomochi, M., Kono, M.: Spatial prisoner's dilemma games with dynamic payoff matrices. *Physical Review E* 65(026112) (2002)
20. Traulsen, A., Claussen, J.C., Hauert, C.: Evolutionary dynamics: from finite to infinite populations. *Phys. Rev. Lett.* 95, 238701 (2005)
21. Turner, P.E., Chao, L.: Escape from prisoner's dilemma in RNA phage 6. *American Naturalist* 161(3), 497–505 (2003)
22. Wooldridge, M.: *An introduction to multiagent systems*. John Wiley and Sons, LTD., Chichester (2002)

Crossover Can Be Constructive When Computing Unique Input Output Sequences

Per Kristian Lehre and Xin Yao

The Centre of Excellence for Research in
Computational Intelligence and Applications (CERCIA),
School of Computer Science, The University of Birmingham,
Edgbaston, Birmingham B15 2TT, United Kingdom
`{P.K.Lehre,X.Yao}@cs.bham.ac.uk`

Abstract. Unique input output (UIO) sequences have important applications in conformance testing of finite state machines (FSMs). Previous experimental and theoretical research has shown that evolutionary algorithms (EAs) can compute UIOs efficiently on many FSM instance classes, but fail on others. However, it has been unclear how and to what degree EA parameter settings influence the runtime on the UIO problem. This paper investigates the choice of acceptance criterion in the (1+1) EA and the use of crossover in the (μ +1) Steady State Genetic Algorithm. It is rigorously proved that changing these parameters can reduce the runtime from exponential to polynomial for some instance classes.

1 Introduction

Evolutionary Algorithms (EAs) are general purpose optimisation algorithms. In principle, they can be applied with little problem domain knowledge, only requiring the user to provide the algorithm with a set of candidate solutions and a way of measuring the quality of each candidate solution. This generality allows EAs to be applied in diverse problem domains, as has been documented extensively. In practice, the application of EAs is often not straightforward as it is often necessary to adjust the parameter settings to the problem at hand. Due to a poor understanding in how and why genetic operators influence the search process, this parameter tuning is often expensive.

Theoretical research like runtime analysis will seldom provide optimal parameter settings for specific real world problems. However, it may provide insight into how and why EAs work and sometimes fail. In particular, a theoretical analysis can point out simple general cases where the choice of a genetic operator has a particularly important effect on the runtime. Equipped with an understanding of how EAs behave in such archetypical cases, a practitioner will be better equipped in making an informed decision as to how to choose parameter settings on a specific real world problem. This paper analyses rigorously the influence of genetic operators on the problem of computing unique input output (UIO) sequences from finite state machines (FSMs), a computationally hard problem from the software engineering domain [1]. UIOs have important applications in

conformance testing of FSMs [2]. Similarly to other approaches in *search based software engineering* [3], the UIO problem has been reformulated as an optimisation problem and tackled with EAs [4,5]. Experimental results show that EAs can construct UIOs efficiently on some instances. Guo *et al.* compared an evolutionary approach with a random search strategy, and found that the two approaches have similar performance on a small FSM, while the evolutionary approach outperforms random search on a larger FSM [5]. Derderian *et al.* presented an alternative evolutionary approach [4], confirming Guo *et al.*'s results.

Theoretical results confirm that EAs can outperform random search on the UIO problem [1]. The expected running time of (1+1) EA on a counting FSM instance class is $O(n \log n)$, while random search needs exponential time [1]. The UIO problem is NP-hard [2], so one can expect that there exist EA-hard instance classes. It has been proved that a combination lock FSM is hard for the (1+1) EA [1]. To reliably apply EAs to the UIO problem, it is necessary to distinguish easy from hard instances. Theoretical results indicate that there is no sharp boundary between these categories in terms of runtime. For any polynomial n^k , there exist UIO instance classes where the (1+1) EA has running time $\Theta(n^k)$ [1].

Do EA settings have any significant impact on the chance of finding UIOs efficiently? Guo *et al.* hypothesise that crossover is helpful [6], without giving further evidence than two example sequences that recombine into a UIO. The theoretical results in this paper confirms this hypothesis, proving that crossover can be essential for finding the UIO in polynomial time. The results also show how modifying the acceptance criterion of an EA can have a similarly drastic impact on the runtime. The remaining of this section provides preliminaries, followed by the main results in Sections 2 and 3.

Definition 1 (Finite State Machine). A finite state machine (FSM) M is a quintuple $M = (I, O, S, \delta, \lambda)$, where I is the set of input symbols, O is the set of output symbols, S is the set of states, $\delta : S \times I \rightarrow S$ is the state transition function and $\lambda : S \times I \rightarrow O$ is the output function.

When receiving an input symbol a , the machine makes the transition from its current state s to a next state $\delta(s, a)$ and outputs symbol $\lambda(s, a)$. The functions λ and δ are generalised to the domain of input sequences in the obvious way.

Definition 2 (Unique Input Output Sequence). A unique input output sequence (UIO) for a state s in an FSM M is a string x over the input alphabet I such that $\lambda(s, x) \neq \lambda(t, x)$ for all states t , $t \neq s$.

To compute UIOs with EAs, candidate solutions are represented as strings over the input alphabet of the FSM, which is here restricted to $I = \{0, 1\}$ [5]. Although the shortest UIOs in the general case can be exponentially long with respect to the number of states [2], all the instances presented here have UIOs of length n . The objective in this paper is to search for an UIO of length n for state s_1 in various FSMs, where the fitness of a input sequence is defined as a function of the *state partition tree* induced by the input sequence [5,1].

Definition 6 (RIDGE FSM). For instance sizes n , $n \geq 2$, define a RIDGE FSM with input and output symbols $I := \{0, 1\}$ and $O := \{a, b\}$ respectively, and $2n$ states $S := Q \cup R$, where $Q := \{q_1, q_2, \dots, q_n\}$ and $R := \{s_1, s_2, \dots, s_n\}$. For all states q_i and s_i , $1 \leq i \leq n$, define the transition and output functions as $\delta(q_i, 0) := q_i$, $\delta(s_i, 1) := s_1$, $\lambda(q_i, 0) := a$, $\lambda(s_i, 1) := a$, and

$$\begin{aligned} \delta(q_i, 1) &:= \begin{cases} s_1 & \text{if } i = n, \\ q_{i+1} & \text{otherwise.} \end{cases} & \delta(s_i, 0) &:= \begin{cases} q_1 & \text{if } i = n, \\ s_{i+1} & \text{otherwise.} \end{cases} \\ \lambda(q_i, 1) &:= \begin{cases} b & \text{if } i = n, \\ a & \text{otherwise.} \end{cases} & \lambda(s_i, 0) &:= \begin{cases} b & \text{if } i = n, \\ a & \text{otherwise.} \end{cases} \end{aligned}$$

The fitness function $\text{UIO}_{\text{RIDGE}, s_1}$ can be expressed as a pseudo-boolean function.

Proposition 1. The fitness function $\text{UIO}_{\text{RIDGE}, s_1}$ associated with the RIDGE FSM instance class of size n takes the values

$$\text{RIDGE}(x) = \begin{cases} 2n - 1 & \text{if } x = 0^n, \text{ and} \\ \sum_{i=1}^n x_i + \sum_{i=1}^n \prod_{j=1}^i (1 - x_i) & \text{otherwise.} \end{cases}$$

Proof. We claim that on inputs x of length n and different from 0^n , the number of states, among the states q_i , $1 \leq i \leq n$, with different output than state s_1 equals $\text{ONEMAX}(x) := \sum_{i=1}^n x_i$ and the number of states, among the states s_i , $2 \leq i \leq n$, with different output than state s_1 equals $\text{LZ}(x) := \sum_{i=1}^n \prod_{j=1}^i (1 - x_i)$. The first claim follows from the characterisation of the easy FSM instance class in [1] (see Proposition 1). All states s_i , $1 \leq i \leq n$, collapse to state s_1 on input 1. Hence, for a state s_i , $2 \leq i \leq n$, if $\lambda(s_1, 0^j 1z) \neq \lambda(s_i, 0^j 1z)$, then $\lambda(s_1, 0^j) \neq \lambda(s_i, 0^j)$. To reach transition (s_n, q_1) from state s_i , it is necessary to have at least $n - i$ 0-bits in the input sequence. Hence, on input 0^j , a state s_j has different output from s_1 if and only if $j > n - i$. The number of states s_i , $2 \leq i \leq n$, with different output from state s_1 on input $0^j 1z$ is j . $\square$

Except for 0^n which is the only UIO of length n for state s_1 , the fitness function is the sum of LZ and ONEMAX. The search points $0^i 1^{n-i}$, $0 \leq i < n$, have identical fitness, forming a neutral path of length $n - 1$. The runtime analysis for RIDGE is similar to that of SPC in [9]. When reaching the path, (1+1) EA will make a random walk until it hits the global optimum. (1+1) EA* will get stuck on the path, only accepting the optimal search point. If the distance to the optimum is large, then it takes long until (1+1) EA* mutates the right bits. The function SPC maximises this distance by embedding an explicitly defined trap. In contrast, RIDGE does not have such an explicitly defined trap. Even without the trap, one can prove that (1+1) EA* is still likely to reach the path far from the optimum because (1+1)* EA optimises ONEMAX quicker than LZ. The formal proof of this, and some of the following theorems have been omitted due to space limitations. (A complete version of this paper containing all the proofs is available as a technical report [10].)

Theorem 1. *The expected time until $(1+1)$ EA finds an UIO of length n for state s_1 in RIDGE FSM using fitness function $\text{UIO}_{\text{RIDGE},s_1}$ is bounded from above by $O(n^3)$.*

Theorem 2. *The probability that $(1+1)^*$ EA has found an UIO of length n for state s_1 in RIDGE FSM using fitness function $\text{UIO}_{\text{RIDGE},s_1}$ in less than $n^{n/24}$ steps, is bounded from above by $e^{-\Omega(n)}$.*

3 Impact of Crossover

Although $(1+1)$ EA is efficient on several instance classes, one can hypothesise that there exist FSMs for which this EA is too simplistic. In particular, when is it necessary to use crossover and a population in computing UIOs? There exists theoretical evidence that crossover is essential on at least some problems, including several artificial pseudo-boolean functions [11,12]. For the Ising model, a small runtime gap was proven for rings [13], and an exponential runtime gap was proven for trees [14]. Recently, crossover was proven essential on a vertex cover instance [15], but this result depends on an artificially low crossover probability. We present an instance class of the UIO problem and a steady state genetic algorithm where crossover is provably essential. When reducing the crossover probability from any positive constant ($p_c > 0$) to no crossover ($p_c = 0$), the runtime increases exponentially. The proof idea is to construct a fitness function where the individuals in the population can follow two different paths, each leading to a separate local optimum. The local optima are separated by the maximal Hamming distance. The global optimum is positioned in the middle between these local optima and can be efficiently reached with an appropriate one-point crossover between the local optima. The paths are constructed to make it unlikely that mutation alone will produce the global optimum. It is worth noting that our analysis is based on specific types of crossover and mutation.

Definition 7. *For instance sizes n , $n \geq 2$ and a constant ϵ , $0 < \epsilon < 1$, define a TwoPATHS FSM with input and output symbols $I := \{0, 1\}$ and $O := \{a, b, c\}$ respectively, and $2(n+1)$ states $S = Q \cup R$, where $R := \{s_1, s_2, \dots, s_{n+1}\}$ and*

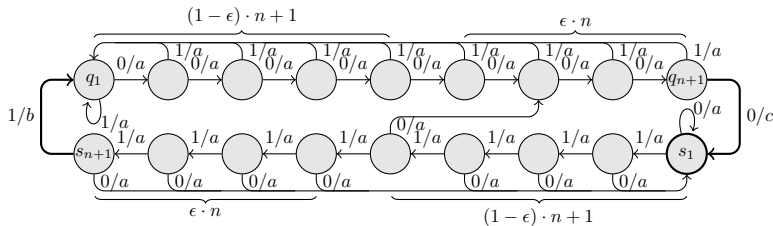


Fig. 2. TwoPATHS FSM instance class

$Q := \{q_1, q_2, \dots, q_{n+1}\}$. The output and transition functions are defined as

$$\begin{aligned} \lambda(q_i, x) &:= \begin{cases} b & \text{if } i = n+1 \text{ and } x = 0, \\ a & \text{otherwise} \end{cases}, & \lambda(s_i, x) &:= \begin{cases} b & \text{if } i = n+1 \text{ and } x = 1, \\ a & \text{otherwise,} \end{cases} \\ \delta(s_i, 0) &:= \begin{cases} q_{(1-\epsilon) \cdot n + 3} & \text{if } i = \epsilon \cdot n + 1, \\ s_1 & \text{otherwise.} \end{cases}, & \delta(s_i, 1) &:= \begin{cases} q_1 & \text{if } i = n+1, \\ s_{i+1} & \text{otherwise.} \end{cases} \\ \delta(q_i, 1) &:= q_1, \quad \text{and,} & \delta(q_i, 0) &:= \begin{cases} s_1 & \text{if } i = n+1, \text{ and} \\ q_{i+1} & \text{otherwise.} \end{cases} \end{aligned}$$

Proposition 2. Let ϵ be any constant $0 < \epsilon < 1$. On input sequences of length n , the fitness function $\text{UIO}_{\text{TWO PATHS}, s_1}$ takes the following values, where $A = \{1^i 0^{\epsilon n} \alpha \mid \alpha \in \{0, 1\}^{(1-\epsilon)n-i}\}$,

$$\begin{aligned} \text{TWO PATHS}(x) &= \begin{cases} 2n+1 & \text{if } x = x^* \\ \text{Lo}(x) + 1 & \text{if } x \in A \setminus \{x^*\} \\ \text{Lo}(x) & \text{if } x_1 = 1 \text{ and } x \notin A \\ \text{Lz}(x) + 1 & \text{if } x_1 = 0 \text{ and } \text{Lz}(x) \geq \epsilon n, \\ \text{Lz}(x) & \text{if } x_1 = 0 \text{ and } \text{Lz}(x) < \epsilon n, \end{cases} & \begin{aligned} \text{Lo}(x) &:= \sum_{i=1}^n \prod_{j=1}^i x_j, \\ \text{Lz}(x) &:= \sum_{i=1}^n \prod_{j=1}^i (1-x_j) \\ x^* &:= 1^{(1-\epsilon) \cdot n} 0^{\epsilon \cdot n}. \end{aligned} \end{aligned}$$

Proof. The states s_{n+1} and q_{n+1} are called *distinguishing* because they have unique input/output behaviours, whereas all other states output a on any input symbol. Clearly, for any two states s and t and input sequence x , if neither state s nor state t reach any distinguishing state on input sequence x , then $\lambda(s, x) = \lambda(t, x) = a^{\ell(x)}$.

On input sequences x of length n , we first claim that any state $s_i \in R$ reaches the distinguishing transition (q_{n+1}, s_1) if and only if the input sequence is on the form $x = 1^{(1-\epsilon) \cdot n + 1 - i} 0^{\epsilon \cdot n} \alpha$. Consider first input sequences of length n on the form $x = 1^j 0 \alpha$ where $j \neq (1-\epsilon) \cdot n + 1 - i$. If $0 \leq j < (1-\epsilon) \cdot n + 1 - i$, then $\delta(s_i, 1^j 0) = s_1$, and from state s_1 , it is impossible to reach state q_{n+1} with the remaining bits α which by assumption must be shorter than n . On the other hand, if $j > (1-\epsilon) \cdot n + 1 - i$, then on input 1^j , we reach a state beyond $s_{(1-\epsilon) \cdot n + 1}$ from which the shortest distance to state q_{n+1} is longer than n . Consider next input sequences of length n on the form $x = 1^{(1-\epsilon) \cdot n + 1 - i} 0^j 1 \alpha$ with $0 \leq j < \epsilon \cdot n$, then $\delta(s_i, 1^{(1-\epsilon) \cdot n + 1 - i} 0^j 1) = q_1$, and it is impossible to reach state q_{n+1} from state q_1 with the remaining substring α which is shorter than n . Our first claim holds, and hence, on input sequence $x^* = 1^{(1-\epsilon) \cdot n} 0^{\epsilon \cdot n}$, only state s_1 among states R reaches the distinguishing transition, and none of the states in Q reaches the distinguishing transition. This implies that this input sequence is a UIO and $\text{TWO PATHS}(1^{(1-\epsilon) \cdot n} 0^{\epsilon \cdot n}) = 2n + 1$.

We secondly claim that $\lambda(s_1, 0^j 1 z) = \lambda(q_i, 0^j 1 z)$ if and only if $\lambda(s_1, 0^j) = \lambda(q_i, 0^j)$ for any state $q_i \in Q$, and $\ell(z) = n - j - 1$ and $1 \leq j \leq n - 1$. ($\implies$) The assumption $\lambda(s_1, 0^j 1 z) = \lambda(q_i, 0^j 1 z)$ implies $\lambda(s_1, 0^j) = \lambda(q_i, 0^j)$ by Lemma 1. ($\impliedby$) The assumption $\lambda(q_i, 0^j) = \lambda(s_1, 0^j) = a^j$ implies that $\delta(q_i, 0^j 1) = q_1$.

Neither state q_1 nor state $\delta(s_1, 0^j 1) = s_2$ reach any of the distinguishing states on input z , hence $\lambda(s_2, z) = \lambda(q_1, z)$, and $\lambda(s_1, 0^j 1z) = \lambda(q_i, 0^j 1z)$.

On input 0^j , a state $q_i \in Q$ has different output from state s_1 if and only if $j > n + 1 - i$. Hence, on input sequences $0^j 1z$, the number of states in Q with different output than state s_1 equals $j = \text{Lz}(0^j 1z)$. Furthermore, by the first claim, the number of states in R with different output than state s_1 on input $0^j 1z$ is at most 1. Therefore $\text{Lz}(0z) \leq \text{TwoPATHS}(0z) \leq \text{Lz}(0z) + 1$. On input symbol 1, all states $q \in Q$ collapse into state q_1 , therefore none of these states will reach a distinguishing state on any input sequence $1z \neq x^*$ of length n . Hence, using a similar argument as for input sequences $0z$ above, we have $\text{Lo}(1z) \leq \text{TwoPATHS}(1z) \leq \text{Lo}(1z) + 1$, which completes the proof. $\square$

If all individuals reach the same local optimum, then the crossover operator will not be helpful. An essential challenge with the idea behind **TWOPATHS** is to ensure that both local optima are reached. In addition to a large population size, some sort of diversity mechanism might be helpful. Here, we will consider a steady state GA where population diversity is ensured through the acceptance criteria.

Definition 8 (($\mu+1$) SSGA)

Sample a population P of μ points u.a.r. from $\{0, 1\}^n$.

repeat

with probability $p_c(n)$,

 Sample x and y u.a.r. from P . $(x', y') := \text{Crossover}(x, y)$.

if $\max\{f(x'), f(y')\} \geq \max\{f(x), f(y)\}$ **then** $x := x'$ and $y := y'$.

otherwise

 Sample x u.a.r. from P . $x' := \text{Mutate}(x)$.

if $f(x') \geq f(x)$ **then** $x := x'$.

$(\mu+1)$ SSGA with crossover probability $p_c = 0$ degenerates into μ parallel runs of the $(1+1)$ EA. The algorithm $(\mu+1)$ SSGA is similar to $(\mu+1)$ RLS introduced in [15], but has a different acceptance criterion. The $(\mu+1)$ RLS accepts both offspring if the best offspring is at least as good as the worst parent, hence allowing the best individual in the population to be replaced with a less fit individual. The $(\mu+1)$ SSGA adopts a more restrictive acceptance criterion, only accepting the offspring if the best offspring is at least as good as the best parent. Each individual in a $(\mu+1)$ SSGA population can be associated with a lineage which interacts little with other lineages, thus facilitating the runtime analysis.

Definition 9 (SSGA Lineage). *If x was added to the population by mutating y , then y is the parent of x . If $z = \alpha_1 \cdot \beta_2$ was added to the population via crossover between $x = \alpha_1 \cdot \alpha_2$ and $y = \beta_1 \cdot \beta_2$, then y is the parent of z if $\alpha_1 = \beta_1$, otherwise x is the parent of z . The lineage of an individual in the population, is the sequence of search point associated with the parent relations.*

Definition 10 (TWO PATHS suffix). *If a search point $x = x_1 \cdots x_i x_{i+1} \cdots x_n$ satisfies $x_1 = x_2 = \cdots = x_i$ and $x_i \neq x_{i+1}$, then the substring $x_{i+1} \cdots x_n$ is called the suffix of search point x .*

Proposition 3. *The probability that any of the initial e^{cn} generations of $(\mu+1)$ SSGA on TWOPATHS contain a non-optimal individual with the bitstring $0^{\epsilon n}$ in its suffix is exponentially small $e^{-\Omega(n)}$.*

Lemma 2. *As long as no individual in the population has a suffix containing substring $0^{\epsilon n}$, the fitness along any lineage of SSGA on TWOPATHS is monotonically increasing.*

To show that the population will be relatively evenly distributed between the two local optima, it is sufficient to prove that there is a constant probability that a lineage will always stay on the same path as it started.

Lemma 3. *For $n \geq 4$, and any lineage x , let t be the generation at which x reaches a local optimum. If no individual until generation t has $0^{\epsilon n}$ in its suffix, then the probability that lineage x reached the local optimum without accepting a search point in which the first bit has been flipped, is bounded from below by $1/12$.*

Theorem 3. *The expected runtime of $(\mu+1)$ SSGA with a constant crossover probability $p_c > 0$ on TWOPATHS is $O(n^2 \mu \log \mu + \exp(n \ln n - \mu/96))$.*

Proof. The process is divided into two phases. Phase 1 begins with the initial population and ends when all individuals have reached either 0^n or 1^n . Phase 2 lasts until the optimum is found. *Phase 1:* We consider a *failure* in phase 1 to occur if at any time during phase 1, there exists an individual with a suffix containing the string $0^{\epsilon n}$. Assume that a failure does not occur. Let ℓ be the lowest fitness in the population, and i the number of individuals with fitness ℓ . In order to decrease the number of individuals with fitness ℓ , it suffices to make a mutation step, select one among i individuals with fitness ℓ , and flip none but the left-most 1-bit (or 0-bit), an event which happens with probability at least $(1-p_c) \cdot (i/\mu) \cdot (1/n) \cdot (1-1/n)^{n-1} \geq (1-p_c)i/e\mu n$. By Lemma 2, the fitness does not decrease along any lineage. Hence, the expected time until the entire population has reached either 0^n or 1^n is bounded from above by $\sum_{\ell=1}^{n-1} \sum_{i=1}^{\mu} e\mu n/i(1-p_c) = O(n^2 \mu \log \mu/(1-p_c))$. By Proposition 3, the failure probability during phase 1 is $e^{-\Omega(n)}$. If a failure occurs, then the number of leading 1-bits can potentially be reduced. Assume pessimistically that $\text{LO}(x) + \text{LO}(x) = 1$ in any lineage x , i.e. the phase restarts. The expected duration of phase 1, then becomes $O(n^2 \mu \log \mu/(1-p_c))/(1-e^{-\Omega(n)}) = O(n^2 \mu \log \mu/(1-p_c))$. *Phase 2:* We consider a *failure* to occur in phase 2, if the phase starts with less than $\mu/64$ individuals on the local optimum with fewest individuals. By Lemma 3, the probability that any lineage has a leading 1-bit (or 0-bit) and never changes path before reaching a local optimum is at least $1/12$. Hence, by Chernoff bounds, the probability that the population contains less than $\mu/24$ individuals which starts with a 1-bit (or 0-bit) and does not change path is bounded from above by $e^{-\mu/96}$. Assuming no failure, the probability of making a crossover step, select two parent individuals 1^n and 0^n , and then making a crossover at point ϵn in

any generation in Phase 2 is at least $p_c(1/24)(23/24)/n$. Hence, the expected duration of Phase 2, assuming no failure is $O(n/p_c)$. If a failure occurs in Phase 2, the optimum can be generated from any search point by mutating at most n bits in any individual, an event which happens in less than n^n expected time.

The unconditional expected duration of Phase 1 and Phase 2 is therefore bounded by $O(n^2\mu \log \mu/(1-p_c) + n/p_c + e^{-\mu/96} \cdot n^n/(1-p_c)) = O(n^2\mu \log \mu/(1-p_c) + n/p_c + \exp(n \ln(n/(1-p_c)) - \mu/96))$. $\square$

Finally, we state the runtime with crossover probability $p_c = 0$. The proof idea is to focus on a single lineage, since the lineages are independent, and distinguish between two conditions. If the lineage has at least ϵn leading 0-bits, then all these must be flipped into 1-bits. If there is at least one 1-bit among the first ϵn bits, then with high probability, a large number of 1-bits must be flipped in the tail of the search point.

Theorem 4. *The probability that $(\mu+1)$ SSGA with crossover probability $p_c = 0$ and population size $\mu = \text{poly}(n)$ finds the optimum of TWOPATHS within 2^{cn} generations is bounded from above by $e^{-\Omega(n)}$, where c is a constant.*

4 Conclusion

This paper has investigated the impact of the acceptance criterion in $(1+1)$ EA and the crossover operator in $(\mu+1)$ SSGA when computing UIOs from FSMs. The objective is to identify simple, archetypical cases where these EA parameter settings have a particularly strong effect on the runtime of the algorithm. The *first part* describes the RIDGE FSM instance class which induces a search space with a neutral path of equally fit search points. Runtime analysis shows that the variant of $(1+1)$ EA which only accepts strictly better search points will get stuck on the path, while the standard $(1+1)$ EA which also accepts equally fit search points will find the UIO in polynomial time. This result shows that apparently minor modification of an algorithm can have an exponentially large runtime impact when computing UIOs. The *second part* considers the impact of crossover when computing UIOs with the $(\mu+1)$ SSGA. The result shows that on the TWOPATHS FSM instance class, the SSGA finds the UIO in polynomial time as long as the crossover probability is a non-zero constant and the population is sufficiently large. However, with crossover probability 0, the runtime of $(\mu+1)$ SSGA increases exponentially. This result means that when computing UIOs, the crossover operator can be essential, and simple EAs including the $(1+1)$ EA can be inefficient. This result is important because although the crossover operator is often thought to be important for GAs, there exist very few theoretical results in non-artificial problem domains confirming that this is the case.

Acknowledgements. The authors would like to thank Pietro Oliveto for useful comments. This work was supported by EPSRC under grant no. EP/C520696/1.

References

1. Lehre, P.K., Yao, X.: Runtime analysis of (1+1) EA on computing unique input output sequences. In: *Proceedings of 2007 IEEE Congress on Evolutionary Computation (CEC 2007)*, pp. 1882–1889 (2007)
2. Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines—a survey. *Proceedings of the IEEE* 84(8), 1090–1123 (1996)
3. Clark, J.A., Dolado, J.J., Harman, M., Hierons, R.M., Jones, B., Lumkin, M., Mitchell, B., Mancoridis, S., Rees, K., Roper, M., Shepperd, M.: Reformulating software engineering as a search problem. *IEE Proceedings-Software* 150(3), 161–175 (2003)
4. Derderian, K.A., Hierons, R.M., Harman, M., Guo, Q.: Automated unique input output sequence generation for conformance testing of fsm's. *The Computer Journal* 49(3), 331–344 (2006)
5. Guo, Q., Hierons, R.M., Harman, M., Derderian, K.A.: Computing unique input/output sequences using genetic algorithms. In: Petrenko, A., Ulrich, A. (eds.) *FATES 2003. LNCS*, vol. 2931, pp. 164–177. Springer, Heidelberg (2004)
6. Guo, Q., Hierons, R.M., Harman, M., Derderian, K.A.: Constructing multiple unique input/output sequences using metaheuristic optimisation techniques. *IEE Proceedings Software* 152(3), 127–140 (2005)
7. Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science* 276, 51–81 (2002)
8. He, J., Yao, X.: A study of drift analysis for estimating computation time of evolutionary algorithms. *Natural Computing* 3(1), 21–35 (2004)
9. Jansen, T., Wegener, I.: Evolutionary algorithms - how to cope with plateaus of constant fitness and when to reject strings of the same fitness. *IEEE Transactions on Evolutionary Computation* 5(6), 589–599 (2001)
10. Lehre, P.K., Yao, X.: Crossover can be constructive when computing unique input output sequences. Technical Report (CSR-08-08), University of Birmingham, School of Computer Science (2008)
11. Jansen, T., Wegener, I.: The analysis of evolutionary algorithms - a proof that crossover really can help. *Algorithmica* 34(1), 47–66 (2002)
12. Storch, T., Wegener, I.: Real royal road functions for constant population size. *Theoretical Computer Science* 320(1), 123–134 (2004)
13. Fischer, S., Wegener, I.: The one-dimensional ising model: Mutation versus recombination. *Theoretical Computer Science* 344(2-3), 208–225 (2005)
14. Sudholt, D.: Crossover is provably essential for the ising model on trees. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2005)*, pp. 1161–1167 (2005)
15. Oliveto, P., He, J., Yao, X.: Analysis of population-based evolutionary algorithms for the vertex cover problem. In: *Proceedings of the IEEE World Congress on Computational Intelligence (WCCI 2008)*, Hong Kong, June 1-6 (2008)
16. Hoeffding, W.: Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association* 58(301), 13–30 (1963)
17. Oliveto, P., Witt, C.: Simplified drift analysis for proving lower bounds in evolutionary computation. Technical Report Reihe CI, No. CI-247/08, SFB 531, Technische Universität Dortmund, Germany (2008)

Power Electronic Circuits Design: A Particle Swarm Optimization Approach^{*}

Jun Zhang^{**}, Yuan Shi, and Zhi-Hui Zhan

Department of Computer Science, Sun Yat-sen University, China, 510275
junzhang@ieee.org

Abstract. The development of power electronics results in a growing need for automatic design and optimization for power electronic circuits (PECs). This paper presents a particle swarm optimization (PSO) approach for the PECs design. The optimization problem is divided into two processes using a decoupled technique and PSO is employed to optimize the values of the circuit components in the power conversion stage (PCS) and the feedback network (FN), respectively. A simple mutation operator is also incorporated into PSO to enhance the population diversity. The algorithm is applied to the optimization of a buck regulator for meeting requirements under large-signal changes and at steady state. Compared with genetic algorithm (GA), PSO can yield more optimized values of circuit components with lower computational effort.

Keywords: Particle swarm optimization (PSO), power electronic circuits (PECs), circuit optimization.

1 Introduction

Power electronics is the technology to efficiently control the electric energy by supplying voltages or currents in a form that is suited for user loads. Power electronic circuits (PECs) have been widely used in various applications, such as mobile devices, computers, televisions and uninterruptible power supply. With the advances in semiconductor technology and electronic packaging as well as the increasing demand in industry, automatic design and optimization of PECs becomes a growing need [1].

The methods for automated synthesis and optimization of circuits are either deterministic or stochastic. Deterministic approaches, like the gradient method and the hill-climbing technique [1] [2], might easily be trapped into local optima, as high nonlinearity always involved in PECs optimization. Also, the performance of some deterministic methods is significantly dependent on the initial starting point of the search. On the other hand, stochastic methods tend to be more suitable for PECs optimization since they explore the search space more extensively. Recently, a class of stochastic methods, called evolutionary algorithms (EAs) has attracted many researchers' attentions due to its high

^{*} This work was supported by NSF of China Project No.60573066 and the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry, P.R. China.

^{**} Corresponding author.

global search capability. EAs are population-based search techniques inspired by natural mechanisms or phenomenon. During the last several decades, many EAs, such as genetic algorithms (GAs) [3] and particle swarm optimization (PSO) [4] have been proposed and applied to plenty of real-world optimization problems.

GAs perform search by simulating the natural evolutionary process. They adopt three evolutionary operations, namely selection, crossover and mutation. A variety of GAs-based design schemes have been used in analog circuits, like transconductance amplifier [5], voltage reference circuit [6], loaded wire antenna [7], etc. Recently, GAs have been successfully applied in the optimization of PECs [8]. Although the optimized results demonstrated the effectiveness of the GAs-based methods, more effective and efficient approaches are still required for the optimization of PECs.

Different from GAs, PSO solves optimization problems by emulating the flocking behavior of birds. It employs a swarm of particles, which represent feasible solutions, to search in a cooperative manner. Each particle has its position and velocity, which are iteratively updated by tracing the personal best position found so far and the global best position found so far. PSO has also been used in analog circuit synthesis, such as RF circuit design [9], microwave filter design [10] and on-chip spiral inductor synthesis [11]. Compared with GA, PSO is easier to implement because of its simple concept. Moreover, PSO converges faster, which can considerably reduce the computational time.

In this paper, a PSO approach for the optimization of PECs is presented. A PEC is decoupled into two parts and particles are employed to find optimal values for the components of each part, aiming to satisfy the specified static and dynamic requirements. To improve the population diversity, a mutation operator is incorporated into the PSO. The algorithm is illustrated with the design of a buck regulator. The optimized results are compared with those obtained by the GA approach [8], showing that PSO is a promising approach for design and optimization of PECs.

The rest of this paper is organized as follows. Section 2 describes the decoupled structure of PECs. Section 3 gives a brief introduction of PSO. Section 4 shows the details of the algorithm for PECs optimization. The design of a buck regulator is illustrated in Section 5 and the conclusion is represented in Section 6.

2 Decoupled Structure of PECs

Fig. 1 shows the basic block diagram of a PEC, in which the circuit is decoupled into two parts, namely the power conversion stage (PCS) and the feedback network (FN).

PCS consists of I_P resistors, J_P inductors and K_P capacitors, which transfers the power from the input source to the output load. FN consists of I_F resistors, J_F inductors and K_F capacitors, which keeps the output voltage v_o to a reference voltage v_{ref} .

The goal of the optimization is to find proper values for these circuit components, aiming to satisfy the static and dynamic responses. Although performing a whole-circuit optimization is feasible, it is computationally intensive since the number of variables is considerably large. In [8], a decoupled technique based on the circuit structure was proposed, which divided the optimization into two processes. One process optimized the component values in PCS and another process tackled FN. This decoupled technique is effective to reduce the computational effort and is adopted in this paper.

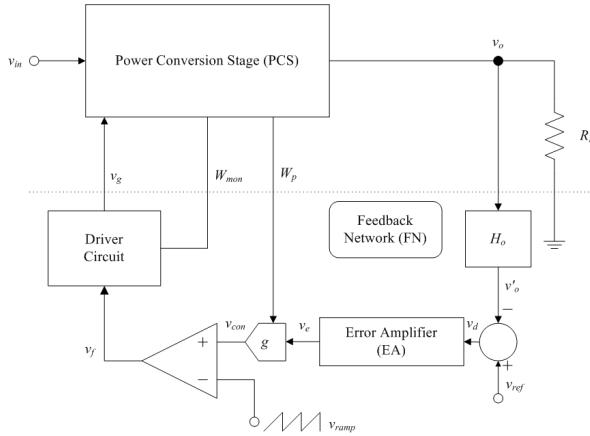


Fig. 1. Block diagram of power electronics circuits

3 Particle Swarm Optimization

PSO is an iterative approach that uses a swarm of M particles. Each particle has its own position and velocity that are vectors with N dimensions. The position of a particle represents a feasible solution in multidimensional search space, and its quality is evaluated by the fitness function. In addition, every particle memories two positions: the personal best position found by the particle so far (denoted as $pbest$) and the global best position found by the whole population so far (denoted as $gbest$). Thus, in PSO, the i th ($i = 1, 2, \dots, M$) particle is represented by three vectors:

The current position: $x_i = (x_{i1}, x_{i2}, \dots, x_{iN})$

The current velocity: $v_i = (v_{i1}, v_{i2}, \dots, v_{iN})$

The personal best position: $pbest_i = (pbest_{i1}, pbest_{i2}, \dots, pbest_{iN})$

In every iteration, the velocity and position of each particle are updated following the equation below:

$$v_{ij}(t+1) = wv_{ij}(t) + c_1r_1(pbest_{ij}(t) - x_{ij}(t)) + c_2r_2(gbest_{ij}(t) - x_{ij}(t)) \quad (1)$$

where t is the iteration index, i is the particle index, and j is the dimension index. r_1 and r_2 are two random values generated with uniform probability from $[0,1]$. Parameter $w > 0$ is called the inertia weight, which determines the impact of the previous velocity: a large inertia weight favors exploration while a small value favors exploitation. Parameters c_1 and c_2 control the influence of the personal best position and the global best position, respectively.

Typically, the velocity is further restricted by a maximum velocity V_{max} . Mathematically, the restriction rule is expressed by:

$$v_{ij}(t+1) = \begin{cases} v_{ij}(t+1), & -V_{max,j} < v_{ij}(t+1) < V_{max,j} \\ V_{max,j}, & v_{ij}(t+1) > V_{max,j} \\ -V_{max,j}, & v_{ij}(t+1) < -V_{max,j} \end{cases} \quad (2)$$

where $V_{\max,j}$ stands for the j th dimension of $V_{\max}$. Assume the search range of the j th variable is $[l_j, u_j]$, $V_{\max,j}$ can be set by:

$$V_{\max,j} = \alpha \cdot (u_j - l_j) \quad (3)$$

where α is a positive parameter.

After the velocity of the i th particle is updated, the particle moves to a new position according to the following equation:

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \quad (4)$$

Particles' positions and velocities are updated repeatedly according to (1), (2) and (4) until meeting the terminal condition. Usually, the optimization process is terminated when the number of iterations reaches the maximum number MAX_ITER .

PSO has been applied to a wide range of optimization problems due to its simple concept, easy implementation and fast convergence. However, PSO is sometimes found to be converged on local optima, the factor of which is the lack of the population diversity [12]. Therefore, incorporating an additional mutation operator can help prevent premature convergence, and thus enhance the algorithm's global search characteristics [13].

4 PSO for PECs Optimization

Based on the decoupled technique, PSO is employed, respectively, for the optimization of PCS and FN. The position of each particle is coded as a vector containing the component values.

4.1 Fitness Function

The fitness function is used to evaluate the quality of a particle's position. Here, we adopt the fitness function in [8] which is based on a time-domain simulation of PECs. For PCS, the fitness function is defined considering the following objectives with each one expressed by an objective function (OF_x).

- 1) steady-state error of the output voltage – OF_1
- 2) operation constraints of the circuit components – OF_2
- 3) steady-state ripple voltage on the output – OF_3
- 4) intrinsic factors concerning with the components – OF_4

Hence, the fitness function for PCS is defined as:

$$\Phi_p(x_{PCS}) = \sum_{R_L=R_{L,min}, \delta R_L}^{R_{L,max}} \sum_{v_{in}=V_{in,min}, \delta v_{in}}^{V_{in,max}} [OF_1(R_L, v_{in}, x_{PCS}) + OF_2(R_L, v_{in}, x_{PCS}) + OF_3(R_L, v_{in}, x_{PCS}) + OF_4(R_L, v_{in}, x_{PCS})] \quad (5)$$

where x_{PCS} represents the position vector for PCS.

For FN, the fitness function is defined based on the following considerations:

- 1) steady state error of output – OF_5
- 2) maximum overshoot and undershoot, damping ratio, and the settling time – OF_6
- 3) steady-state ripple voltage on output – OF_7

4) dynamic behaviors during the large-signal change – OF_8

Thus, the fitness function for FN is defined as:

$$\Phi_F(x_{FN}) = \sum_{R_L=R_{L,min}, \delta R_L}^{R_{L,max}} \sum_{V_{in}=V_{in,min}, \delta V_{in}}^{V_{in,max}} [OF_5(R_L, v_{in}, x_{FN}) + OF_6(R_L, v_{in}, x_{FN}) + OF_7(R_L, v_{in}, x_{FN}) + OF_8(R_L, v_{in}, x_{FN})] \quad (6)$$

where x_{FN} represents the position vector for FN.

4.2 Steps of Optimization

The optimization processes for PCS and FN are similar except for the definition of the fitness function. The steps of optimization are illustrated as follows and the flowchart of the algorithm is provided in Fig. 2.

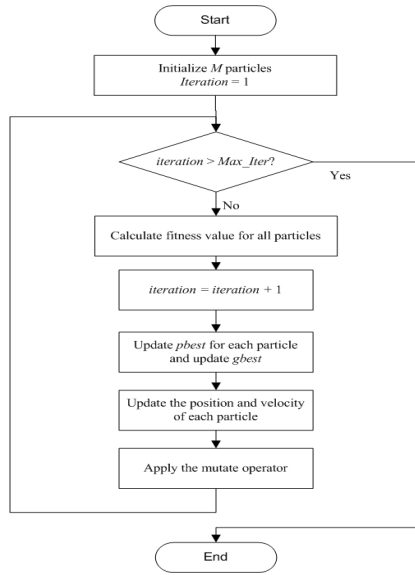


Fig. 2. Flowchart of PSO for PECs optimization

Step 1) Initialize a number of M particles. For the j th circuit component, randomly generate a feasible value within the user-defined search range of $[l_j, u_j]$. The j th velocity is assigned by a random value in the range of $[-V_{\max,j}, V_{\max,j}]$.

Step 2) Calculate the fitness value for each particle. Use the fitness function (5) for PCS and (6) for FN, respectively.

Step 3) Update the personal best position ($pbest$) for each particle according to its fitness value. Update the global best position ($gbest$) for the whole population.

Step 4) Update the position and velocity of every particle.

Step 5) Apply the mutation operator to enhance the population diversity. The operator is performed as follows. For every dimension of each particle, a random value

$r \in [0,1]$ is generated and is compared with the predetermined mutation probability P_m . If r is smaller than P_m , the corresponding component value is randomly reinitialized.

Step 6) If the number of iterations exceeds a maximum value MAX_ITER , the optimization process will be ended, otherwise go to Step 2).

5 Design Example

In this section, the above algorithm is illustrated with the design of a buck regulator with overcurrent protection [14] as shown in Fig. 3. The required specifications are listed in Table 1. For the optimization of PCS, L and C are the designed components and R_L , r_C , r_E are assumed to be known. For FN, all component values are required to be optimized.

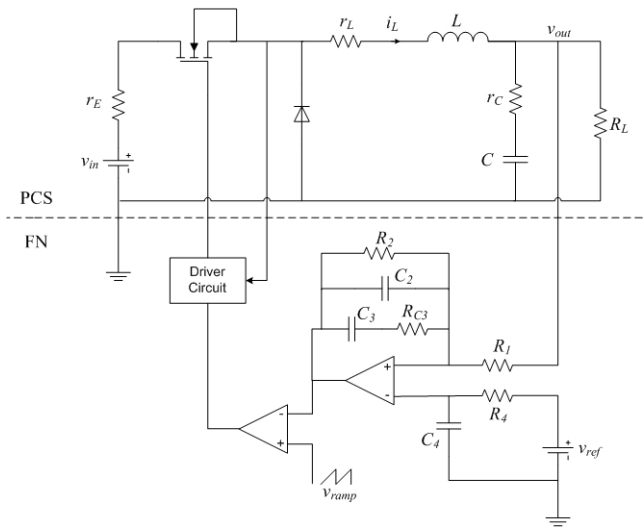


Fig. 3. Schematic of a buck regulator with over current protection

Table 1. Required specifications

Attribute	Value
Input voltage range	40V±20V
Output load range	5Ω~10Ω
Nominal output voltage	5V±1%
Switching frequency	20kHz
Maximum setting time	20ms

The parameter setting for PSO is given in Table 2, in which w is decreased linearly from 1.2 to 0.6 though the optimization process. As suggested in [15], this strategy well balances the exploration and exploitation of the search.

Table 2. Parameter setting for PSO

Parameter	Value
<i>POPSIZE</i>	30
<i>MAX_ITER</i>	500
<i>w</i>	1.2→0.6
<i>c</i> ₁	2.0
<i>c</i> ₂	2.0
<i>α</i>	0.5
<i>P_m</i>	0.02

The fitness value versus the number of iterations in the optimization of FN is shown in Fig. 4. Compared with the related figure given in [8] which was optimized by a GA (the optimizing curve of GA is not presented here, refer to Fig. 7(b) in [8]), PSO converges much faster and achieves significantly higher fitness value. Apparently, using PSO for PECs optimization can considerably lower the computational effort.

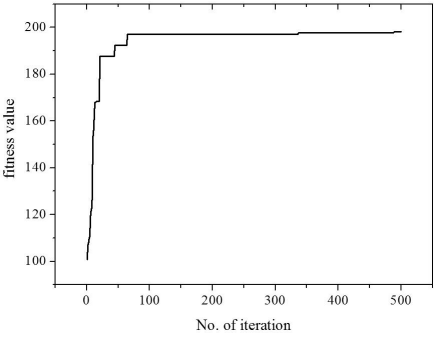


Fig. 4. Fitness values versus the number of iterations in the optimization of FN

Table 3. Optimized results after 500 iterations

Part	Parameter	Value
PCS	<i>L</i>	200μH
	<i>C</i>	100μF
FN	<i>R</i> _{C3}	470Ω
	<i>C</i> ₂	0.2μF
	<i>C</i> ₃	5.56μF
	<i>R</i> ₂	3MΩ
	<i>C</i> ₄	0.18μF
	<i>R</i> ₄	7.644kΩ
	<i>R</i> ₁	60Ω

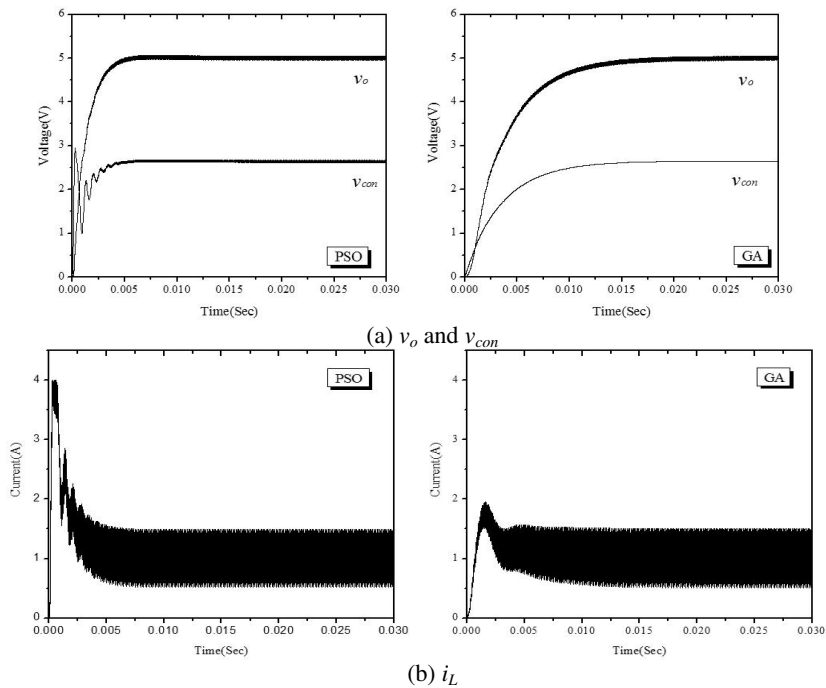


Fig. 5. Simulated startup transient responses when v_{in} is 20V and R_L is 5Ω

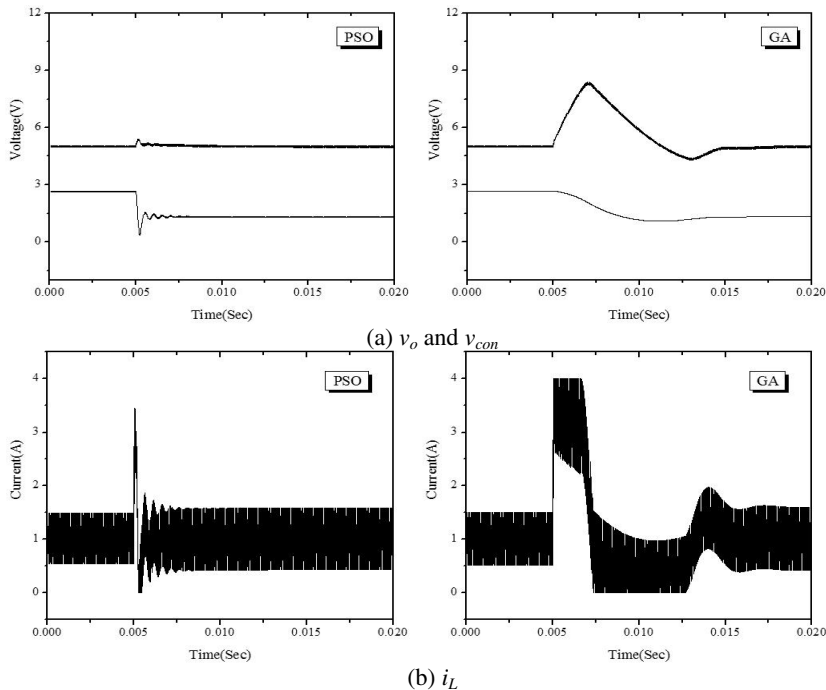


Fig. 6. Simulated transient responses when v_{in} is changed from 20V to 40V and R_L is 5Ω

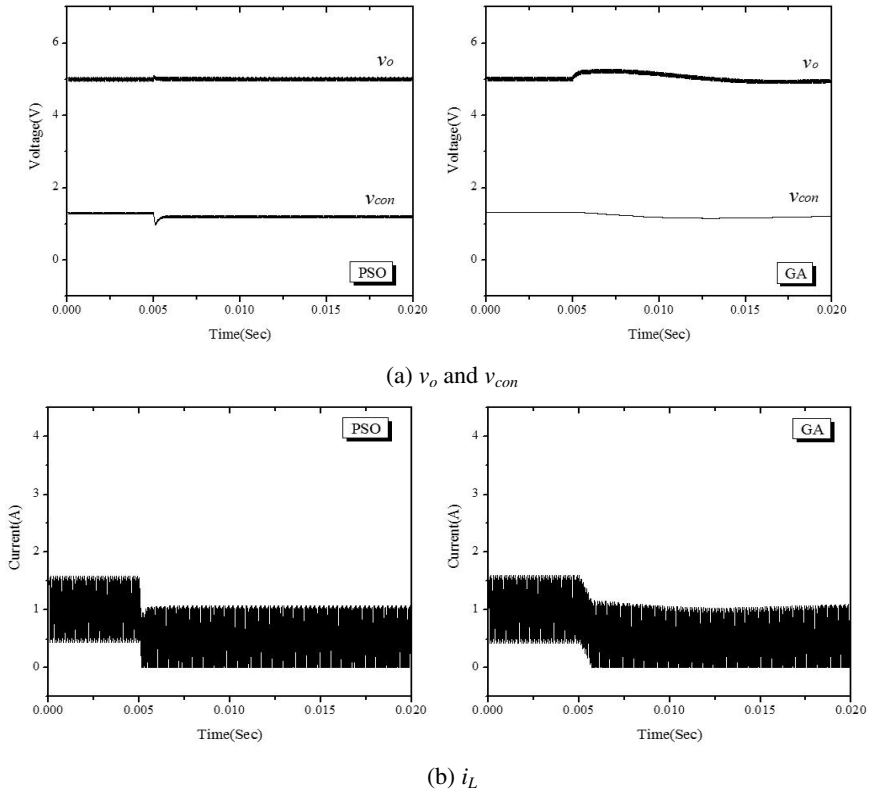


Fig. 7. Simulated transient responses when R_L is changed from 5Ω to 10Ω and v_{in} is 40V

Table 3 gives the optimized values of PCS and FN obtained by the proposed PSO. Fig. 5 shows the simulated startup transients, when the input voltage is 20V and the output load is 5Ω . The setting time of the circuit optimized by PSO is less than 7ms, which is shorter than that in [8]. Fig. 6 illustrates the simulated transient responses when the input voltage is changed from 20V to 40V. Fig. 7 shows the theoretical transients when the output load is changed from 5Ω to 10Ω . From the figures, it can be seen that, the circuit optimized by PSO has much smaller disturbances and shorter response time than the ones in [8], confirming the advantage of the PSO approach.

6 Conclusions

This paper applies a PSO approach for the design and optimization of PECs. No complicated mathematical analysis of the whole circuit is required in the presented method. A design example on a buck regulator with overcurrent protection demonstrates the three advantages of the PSO approach: easy implementation, fast convergence and high global search capability, which can significantly improve the efficiency and effectiveness of PECs optimization and give highly-optimized values of circuit components.

References

1. Verghese, G.C., Bruzos, C.A., Mahabir, K.N.: Averaged and sampled-data model for current-mode control: A reexamination. In: *Proceedings PESC 1989*, pp. 484–491 (1989)
2. Massara, R.E.: *Optimization Methods in Electronic Circuit Design*. Longman, New York (2000)
3. Holland, J.H.: *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor (1975)
4. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: *Proceedings IEEE Int. Conf. Neural Networks*, vol. 4, pp. 1942–1948 (1995)
5. Dhanwada, N.N., Nunez-Aldana, A., Vemuri, R.: A genetic approach to simultaneous parameter space exploration and constraint transformation in analog synthesis. In: *Proceedings IEEE Int. Sym. Circuits Systs.*, pp. 362–365 (1999)
6. Nam, D., Seo, Y., Park, L., Park, C., Kim, B.: Parameter optimization of a voltage reference circuit using EP. In: *Proceedings IEEE Int. Conf. Evolutionary Computation*, pp. 301–305 (1998)
7. Lee, K.C.: Genetic algorithms based analyses of nonlinearly loaded antenna arrays including mutual coupling effects. *IEEE Trans. on Antennas and Propagation* 5, 776–781 (2003)
8. Zhang, J., Chung, H., Lo, W., Hui, S., Wu, A.: Implementation of a decoupled optimization technique for design of switching regulators using genetic algorithms. *IEEE Trans. Power Electron* 16, 752–763 (2001)
9. Jinho, P., Kiyong, C., David, J.A.: Parasitic-Aware RF Circuit Design and Optimization. *IEEE Trans. on Circuits and Systems-I* 51, 1953–1966 (2004)
10. Wen, W., Yilong, L., Jeffrey, S.F., Yong, Z.: Particle Swarm Optimization and Finite-Element Based Approach for Microwave Filter Design. *IEEE Trans. on Magnetics* 41, 1800–1803 (2005)
11. Sushanta, K., Mandal, S.S., Amit, P.: ANN- and PSO-Based Synthesis of On-Chip Spiral Inductors for RF ICs. *IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems* 27, 188–192 (2008)
12. Ratnaweera, A., Halgamuge, S.K., Watson, H.C.: Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Trans. on Evolutionary Computation* 8(3), 240–255 (2004)
13. Paul, S.A.: An Investigation into Mutation Operators for Particle Swarm Optimization. In: *Proceedings IEEE Congress on Evolutionary Computation*, pp. 1044–1051 (2006)
14. Bedrosian, D., Vlach, J.: Time-domain analysis of networks with internally controlled switches. *IEEE Trans. Circuits Syst. I* 39, 199–212 (1992)
15. Shi, Y.H., Eberhart, R.C.: Parameter selection in particle swarm optimization. In: *Proceedings of the 7th Ann. Conf. on Evolutionary Programming*, San Diego, CA, pp. 591–600 (1998)

Computational Intelligence in Radio Astronomy: Using Computational Intelligence Techniques to Tune Geodesy Models

Daniel Angus^{1,*} and Adam Deller²

¹ The University of Queensland, Queensland Brain Institute
d.angus@uq.edu.au

² Swinburne University of Technology, Centre for Astrophysics and Supercomputing
adeller@astro.swin.edu.au

Abstract. In this paper a number of popular Computational Intelligence (CI) algorithms are used to tune Geodesy models, a radio astronomy problem. Several single and multiple objective variations of the Geodesy problem are examined with good results obtained using state-of-the-art CI algorithms. These novel applications are used to develop insights into methods for applying CI algorithms to unknown problem domains and to provide interesting solutions to the Geodesy models used.

1 Introduction

Astronomy, one of the oldest of the sciences has many important problems with far reaching implications. Radio astronomy, which involves the observation of radiation usually greater than one millimetre is important in observing particular celestial objects, such as pulsars. To observe such phenomena, radio astronomers may make use of large scale, spatially separated radio antenna arrays, a technique known as Very Long Baseline Interferometry (VLBI). The use of such equipment comes with certain restrictions though due to factors such as random atmospheric noise and fundamental equipment limitations. Because of this, methods to limit the contribution of noise and errors and thus enhance the quality of observations are important.

One potential source of error in observation relates to the precise location of the actual observational antenna stations. Due to tectonic plate movement and other forces the positions of these antennae are in constant motion. Geodesy is a field which deals directly with measuring and studying the earth's geometry thus it is a useful to apply geodesy techniques to the problem of accurately predicting the positions of antennae.

Computational models that describe these position errors are available, however due to the complexity of the system under investigation it is not always known what parameters should be used in such models. Previously, such models

* This authors' research was partly sponsored by Swinburne University of Technology, Complex Intelligent Systems Lab.

relied on simple deterministic solvers checking through vast numbers of parameter combinations to determine the best parameterisation, however it is obvious that such solutions can be computationally expensive.

Computational Intelligence (CI) is a broad term used to describe many different algorithms, classification systems and computational models usually based on observed physical phenomena. Examples of CI techniques include Evolutionary Algorithms, Swarm Intelligence and Artificial Neural Networks. These problem solvers can often be fast and computationally inexpensive as they tend to adjust their search behaviour dynamically during run-time to better exploit promising areas of the search space. Such techniques have been shown to be effective on complex, multi-modal search spaces typical of that of the geodesy problem described above.

In this paper a range of popular CI techniques are used to optimise a geodesy model. Various techniques aimed at improving the efficacy of the results are explored and the final results are shown to be competitive with an alternative, computationally expensive, deterministic solver. The paper begins by explaining the problem in detail in Sec. 2. The algorithms and their specific application strategies are explained in Sec. 3. Results are provided in Sec. 4, and concluding remarks offered in Sec. 5.

2 Problem Details

2.1 Problem Description

The radio astronomy problem described in this section is taken from the field of Very Long Baseline Interferometry (VLBI). The solution of this problem is significant to radio astronomers since an accurate geometric model is essential to correctly align sampled data sequences from different antennas in time for any interferometry experiment. Errors in antenna positions result in a time-variable offset between the data streams, which reduces image fidelity and introduces systematic errors to source position estimates.

Radio interferometry involves the temporal alignment of signals from different radio telescopes (by means of electronic delays) and correlation of the signals for the purpose of determining information about the spatial frequencies of the radio sky within the telescope's field of view. The collected data is later Fourier transformed to create an image of the radio sky. This data stream alignment requires each telescope's geocentric (geographical) position to be precisely determined, and is made more difficult due to the addition of propagation delays through the Earth's atmosphere, time-stamping errors, the source position and structure, and Earth orientation.

The residual (unmodelled) delay between a pair of antennas (a baseline) can be estimated by fitting the correlated signal phase as a function of frequency, a process known as *fringe-fitting*. For typical astronomical observations, the residual delay (usually resulting from unmodelled atmospheric contributions and time-stamp errors) is simply an error that is removed, and its source is immaterial. However, it is possible to use the residual delays to determine errors in antenna

$$\delta_{B-A} = ((\mathbf{B}' \cdot \hat{\mathbf{S}}' - \mathbf{B} \cdot \hat{\mathbf{S}}) - (\mathbf{A}' \cdot \hat{\mathbf{S}}' - \mathbf{A} \cdot \hat{\mathbf{S}}))/c + (C_B - C_A) + (A_A - A_B) + N_{BA} \quad (1)$$

Where:

- $\mathbf{B}, \mathbf{A}$: Geocentric antenna vectors (user defined variables)
- $\hat{\mathbf{S}}$: Unit vector toward the source (user defined)
- $\mathbf{B}', \mathbf{A}'$: Geocentric antenna vectors (observed values)
- $\hat{\mathbf{S}}'$: Unit vector toward the source (observed value)
- c : Speed of light
- C : Time-stamp error at an antenna
- A : Unmodelled delay due to atmospheric propagation
- N_{BA} : Error introduced by finite signal/noise.

positions, a procedure known as *geodesy*. Geodesy is a challenging problem, since attaining millimetre accuracy requires measurement of relative delays on the order of picoseconds, and the removal of all other contaminating effects on the residual delay. The value of residual delay δ from antenna $\mathbf{B}$ to antenna $\mathbf{A}$ obtained from the fringe-fit solutions can be written as Equ. 1.

Of these values, N_{BA} and C have no dependence on antenna or source position. N_{BA} is randomly distributed and C is a smoothly varying function of time with modern frequency standards, leaving the geometric $(\mathbf{A}, \mathbf{B}, \hat{\mathbf{S}})$ and propagation terms (A). In a dedicated geodesic experiment, weather information is monitored at all antennas and used to correct *a priori* atmospheric propagation models, leaving antenna position errors as the dominant contribution to residual delay. However, even if atmospheric and source position errors cannot be completely subtracted from the residual delays, an estimate of antenna position errors can still be made by assuming the atmospheric and source error delay contributions to be uncorrelated with baseline error delay contributions. An example of an observed residual delay for a single baseline tracked over a 24 hour period is presented in Fig. 1¹.

2.2 Objective Function

From an optimisation perspective the interest is in minimising the magnitude of the residual delay per baseline (antenna pair). This is achieved in this case by selecting appropriate values for the vectors $\mathbf{A}, \mathbf{B}$ and $\hat{\mathbf{S}}$. Like any black box optimisation problem though these vectors are unknown and as such trial and error must be used to select their values. These vectors translate into 14 configurable (scalar) parameters per baseline. However, since one antenna is used as a common reference antenna for all antennae in the system the actual number of configurable parameters per antenna is 7. These values represent the x, y, and z positions, two clock offsets (one per observed frequency band), a clock rate (electronic delay) and clock acceleration (drift in timestamp error). The numbers are

¹ A far more rigorous description can be found in [1].

Table 1. Search space range per individual problem dimension for all datasets

Parameter	Lower bound	Upper bound	Range
1,2,3	-1.0	1.0	2.0
4,5	-5.0E-9	5.0E-9	1.0E-8
6,7	-5.0E-8	5.0E-8	1.0E-7

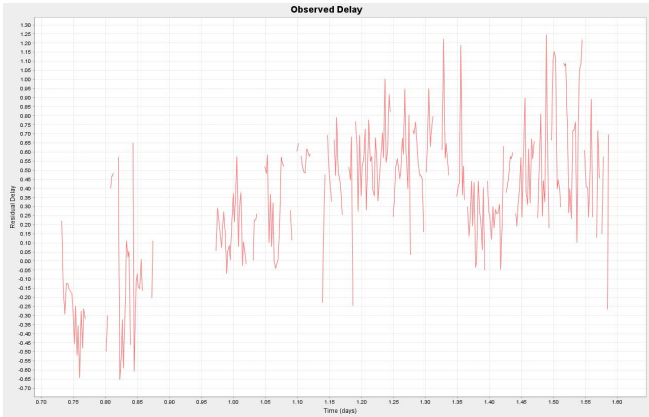


Fig. 1. The observed residual delay for one observing band of an antenna pair (Parkes and ATCA) tracked over a 24 hour period taken from data set STA-131AV

real numbers limited within the range given in Tab. 1 and are valid within this entire range.

The objective value for each baseline is calculated by measuring the difference between each predicted data point (derived from the 14 parameters selected per baseline) and its corresponding observed (real) data point. These differences are combined in an RMS fashion into a single composite objective value. An optimal parameter selection would allow the predicted and observed residual delays to overlap each other perfectly such that if one was subtracted from the other they would essentially cancel each other out, however due to the random noise term N_{BA} this is impossible, and a solution with minimal difference is deemed optimal.

2.3 Problem Data

The problem data used was gathered from five observation points, Parkes, ATCA, Mopra, Hobart and Ceduna. For all data sets Parkes is used as the reference antenna as it is the most sensitive, thus minimising the contribution of the noise term N_{BA} . The data sets STA-131AU and STA-131AV were based on observations of two different sets of radio sources and are available upon request from the Australia Telescope National Facility². Use of these datasets gives rise to four separate baselines that all use Parkes as a common reference.

² Contact: vlbi@atnf.csiro.au

2.4 Multiple and Single Objective Variations

There are a variety of ways to attempt such an optimisation problem. If all four baselines are optimised in parallel, the problem can be phrased as a multiple or single objective optimisation problem. In the multiple objective optimisation (MOO) variant all input parameters (35 in total³) are varied at once, while the objective values for each baseline are treated as independent values to be minimised, denoted as 35D-MO. The single objective optimisation (SOO) variant would require the same number of input parameters, however the objective values can be combined in an RMS fashion to obtain a single objective value, denoted as 35D-SO. Alternatively the seven parameters for the reference antenna can be fixed at zero, resulting in the separation, or decoupling, of the baselines into four separate single objective, 7 dimension, CFO problems, denoted as 7D-SO.

3 Algorithm Description

3.1 Algorithms Used

Multiple objective algorithms used in this study include the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [2] and a Population-based Ant Colony Optimisation algorithm (PACO-MOFO) [3]. The NSGA-II algorithm is a state-of-the-art MOO algorithm which extends many of the basic functions of the canonical Genetic Algorithm. It uses a dominance ranking procedure and crowding distance comparison to determine its population composition, and uses traditional Genetic Algorithm procedures such as crossover and mutation in new solution construction. The PACO-MOFO algorithm is a recently developed MOO algorithm which uses a stepwise solution construction procedure, like traditional Ant Colony Optimisation techniques, to create new solutions. The PACO-MOFO algorithm uses dominance ranking to score solutions and a crowding replacement operation is used to add new solutions to the existing population. These algorithms have been selected due to their success in past studies when applied to complex multiple objective function optimisation problems.

Single objective algorithms include Particle Swarm Optimisation (PSO) [4], Crowding Population-based Ant Colony Optimisation (CPACO) [5], Deterministic Crowding Genetic Algorithm (DCGA) [6,7] and Differential Evolution (DE) [8]. PSO is a optimisation technique inspired by the flocking of birds, it uses a vector-based approach to direct a population of solutions through the search space. The solutions are evaluated periodically to determine whether new interesting optima have been located. CPACO is another Population-based Ant Colony Optimisation algorithm which uses a stepwise solution construction mechanism to construct solutions one dimension at a time. Good solutions are stored in a population and each solution manipulates an artificial ‘pheromone’

³ 7 parameters are specified per antenna (including the reference). Since the same parameters for the reference antenna are used by each baseline this results in 35 dimensions for a 5 antennae problem.

which is used to bias the construction of new solutions. DCGA is a Genetic Algorithm which uses a real-value encoding and a deterministic crowding replacement operation. Two solutions are created at a time from two parent solutions using crossover and mutation operators. The solutions are then compared directly to each of these parents and if either or both of the new solutions are better than their closest matching parent, the parent is replaced. The DE algorithm is similar to PSO as it is also uses a vector based approach to search the given search space. In its simplest form DE selects three population members at random, the difference between two is calculated and used to perturb the third. If the new perturbed solution is better than a selected existing population member that population member is replaced.

3.2 Methodology Employed

Of particular interest in this study is the development of methods to increase the search efficacy of the previously described CI techniques. Given that very little is known about the search space composition and dynamics the development of general purpose solution solving methods is of interest. The particular methodologies that are tested include:

Search Space Size Reduction. It is intuitive that a reduction in the size of the search space will lead to a much more thorough search given a limited number of function evaluations. However as it is not known where the actual global optimal solution is located the reduction of the search space may actually exclude this global optimal solution thus possibly decrease the efficacy of a global search strategy. The search space size is reduced using a first-order solution estimation technique which performs a rough fit of the data and centers the search space around this solution.

Search Space Relocation. Given that a reduction in the size of the search space may lead to the exclusion of a global optimal solution, a possible alternative is to allow the search space to periodically relocate. It is thought that such a relocation may, for some period of time, allow the inclusion of more promising areas of the search space thus increasing the overall search efficacy. To achieve the relocation each algorithm is allowed m function evaluations after which the best solution is used as an origin where the search space is relocated around. The actual size of the search space does not change, simply the minimum and maximum boundaries are shifted. After repositioning the search space boundaries the algorithm is reinitialised and allowed to run again. This process is repeated p times so that overall the algorithm executes $m \times p$ function evaluations. The best solution found during the entire run is reported rather than the last best solution found.

Problem Variations. In the 35D-MO, 35D-SO configurations the problem is treated as a $7n$ dimension problem, where n is the number of antennae, inclusive

of the reference antenna. Alternatively, if the values for the reference antenna are fixed at zero this allows the problem to be decomposed into several smaller (less dimensions) problems since the effect of a variable reference is removed. Removal of the reference antenna's variability is a valid approach since baseline measurements are always relative to this reference antenna anyway, as such it means that all baseline variation is occurring at an individual antenna rather than being split between the reference and selected antennae. By fixing the reference antenna the problem can be treated as four independent 7 dimension problems. The complexity of evaluating the objective value of these variations is dependent on the number of input variables. As a result the 35D-MO, 35D-SO and 7D-SO (evaluated for all four baselines) all require a comparable amount of computation to evaluate solutions.

4 Results and Analysis

4.1 Experiment Setup

The 35D-MO and 35D-SO configurations were tested first, using the default variable ranges defined in Tab. 1. The 35D-SO and 7D-SO configurations were then tested, still using the default variable ranges, to determine the effect of separating the problem into multiple sub-problems⁴. A reduction of the search space size was then tested using only the 7D-SO problem variant. Finally a reduction of the search space size with dynamic search space relocation was performed using the 7D-SO problem variant.

All algorithms were allowed a total of 100,000 function evaluations per trial, and all trials were repeated 100 times using different random seeds. Non-parametric statistics were used to validate significant differences between reported results, specifically the Mann-Whitney rank sum test was used with a confidence level of 95%. The results reported are the mean (with standard deviation) and those highlighted in bold were the best values found. Algorithm parameter configurations were set using the suggested default values in the original references, and with population sizes of 100.

4.2 Experiment Results

Given that the objective space of the 35D-MO variant contains four dimensions, it was difficult to visualise the results obtained by the NSGA-II and PACOMOFO algorithms using traditional visualisation methods such as summary attainment surface comparison. Instead the objective values were combined for each individual solution and the best aggregate value is reported in Tab. 3 to provide values that are directly comparable to the results obtained by the algorithms used in the 35D-SO and 7D-SO configurations reported in Tab. 2.

⁴ For the 7D-SO model the algorithms use four independent 7D objective functions, however, the value reported is obtained by recombining the best solutions' input parameters found for each sub-problem and evaluating this combination using the 35D objective function.

Table 2. Results obtained by the Random, PSO, DE, CPACO and DCGA algorithms, on the 35D-SO and 7D-SO variants using the default search space size

Problem	Random	PSO	DE	CPACO	DCGA
AU(35D-SO)	12.60 (1.44)	5.79 (1.72)	0.76 (0.07)	1.19 (0.16)	2.41 (0.42)
AU(7D-SO)	-	1.16 (0.23)	0.54 (0.00)	0.60 (0.04)	0.59 (0.01)
AV(35D-SO)	12.66 (1.62)	5.71 (1.47)	0.68 (0.07)	1.17 (0.21)	2.30 (0.53)
AV(7D-SO)	-	1.07 (0.30)	0.48 (0.00)	0.53 (0.02)	0.51 (0.01)

Table 3. Results obtained by the NSGA-II and PACO-MOFO algorithms on the 35D-MO variant

Problem (metric)	NSGA-II	PACO-MOFO
AU (combined score)	39.84 (14.02)	8.36 (1.81)
AV (combined score)	43.09 (18.73)	8.89 (2.25)

Table 4. Results obtained by the Random, PSO, DE, CPACO and DCGA algorithms on the 35D-SO and 7D-SO variants using heuristic search space reduction

Problem	Random	PSO	DE	CPACO	DCGA
AU(35D-SO)	6.13 (1.05)	0.65 (0.04)	0.55 (0.00)	0.68 (0.05)	1.30 (0.35)
AU(7D-SO)	-	0.59 (0.00)	0.59 (0.00)	0.59 (0.00)	0.59 (0.00)
AV(35D-SO)	6.27 (1.14)	0.57 (0.04)	0.50 (0.00)	0.62 (0.06)	1.22 (0.36)
AV(7D-SO)	-	0.51 (0.00)	0.51 (0.00)	0.51 (0.00)	0.51 (0.00)

Table 5. Results obtained by the DE and CPACO algorithms on the 7D-SO variant using heuristic search space reduction and dynamic search space re-allocation

Problem	DE	CPACO
AU(35D-SO)	0.60 (0.02)	0.70 (0.13)
AU(7D-SO)	0.52 (0.00)	0.52 (0.00)
AV(35D-SO)	0.56 (0.02)	0.61 (0.07)
AV(7D-SO)	0.48 (0.00)	0.48 (0.00)

Table 6. Positional errors obtained for the STA-131AV dataset using an alternative (but more complex) geodesy model with a deterministic solver. Positional errors obtained for the STA-131AV dataset using DE and CPACO on the 7D-SO problem configuration with the heuristic search space reduction and dynamic relocation technique. Magnitude of difference between the positional errors.

Baseline	Deterministic Solver			DE / CPACO			Difference		
	x (m)	y (m)	z (m)	x (m)	y (m)	z (m)	x (m)	y (m)	z (m)
ATCA	-0.08	0.13	0.31	-0.16	0.14	0.22	0.08	0.01	0.09
Mopra	-0.08	0.08	0.31	-0.15	0.16	0.13	0.07	0.08	0.18
Hobart	0.00	0.18	-0.07	-0.15	0.36	-0.31	0.15	0.18	0.24
Ceduna	-1.72	1.65	-0.89	-1.54	1.47	-0.58	0.18	0.18	0.31

4.3 Experiment Analysis

The results presented in Tab. 2 indicate that when using the default search space range, DE is the best performing algorithm. These results also indicate that when using the default range all algorithms perform better when the reference antenna is fixed at zero and the problem is decoupled into 4 smaller sub-problems (7D-SO). For the Multiple Objective variation (35D-MO) the results (Tab. 3) seem to be quite poor, most likely because these algorithms focus attention across a larger area of the search space (to locate multiple Pareto optimal solutions) when compared to the single objective algorithms. Considering that the problem objectives are easily decomposed into separate sub-problems by fixing the reference antenna at zero, as in 7D-SO, it is unlikely for 35D-MO to be competitive.

When a heuristic search space reduction technique is used, the results for both the 35D-SO and 7D-SO variations for all but the DE algorithm are improved (Tab. 4). The results for DE are interesting because the 35D-SO results are improved using the heuristic search space reduction, however the 7D-SO results are made worse through the reduction of the search space size. This result indicates that the heuristic may exclude optimal areas of the search space. Such a result in the case of DE is bad because this algorithm is able to find good solutions on the larger search space, however in the case of the other algorithms tested such an exclusion is not bad considering these algorithms don't actually find the optimal solution anyway. A similar result was found in a previous CI algorithm study[9]. The optimal search space exclusion is reversed when the search space relocation technique is applied (Tab. 5), and in this case the best results for all experiments performed are obtained by the DE and CPACO algorithms. The agreement between the results obtained by both the DE and CPACO algorithms on the 7D-SO problems using search space reduction and relocation suggests that a global optimal solution has been located.

A deterministic technique operating on a more complex geodesy model was used previously to compute the positional errors for the STA-131AV dataset. This technique used exact measurements of external factors such as the atmospheric noise in its calculations, resulting in a much higher fidelity solution. The results for the positional errors are reported for all four antennas (using Parkes as the reference) and as a comparison the best result obtained with the DE and CPACO algorithms using the 7D-SO variation with heuristic search space size reduction and dynamic relocation (Tab. 6).

Given that the estimated error in the deterministic techniques measurements are approximately $\pm 10\text{cm}$ per dimension, the DE/CPACO results obtained for both ATCA and Mopra are deemed comparable. For the Hobart and Ceduna baselines the DE/CPACO positional errors are larger than the allowed $\pm 10\text{cm}$, which is postulated to be due to unaccounted systemic errors in the geodesy model rather than the optimisation algorithm not obtaining a good solution. These systemic errors could manifest themselves in a variety of ways since the deterministic model uses cleaner radio data, models the atmosphere better thus reducing introduced noise and uses multiple widely spaced frequency bands which results in an increase in sensitivity.

5 Conclusion

This study has shown the usefulness of CI algorithms in yielding quality results on a previously unstudied real-world optimisation problem. The results obtained by the DE and CPACO algorithms were in some cases shown to be within error bounds of previously determined values using a more accurate geodesy model. Perhaps the most informative result obtained related to the effect of heuristics on the accuracy of the result obtained. For PSO, DCGA and CPACO a reduction in the search space size advantaged their search performance, while for DE such a reduction hindered its search performance. Aside from the quantitative results presented, the actual exercise of applying CI algorithms to the problem was able to provide the Astrophysics domain expert with new insight into the underlying problem dynamics. Such insights are hard to quantify and thus are not reported, however this result indicates an important reason as to why one might consider the application of CI algorithms to their problem.

References

1. Fomalont, E.B.: Astrometry and Geodesy. In: Taylor, G.B., Carilli, C.L., Perley, R.A. (eds.) *Synthesis Imaging in Radio Astronomy II*. Astronomical Society of the Pacific Conference Series, vol. 180, p. 463 (1999)
2. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In: Deb, K., Rudolph, G., Lutton, E., Merelo, J.J., Schoenauer, M., Schwefel, H.-P., Yao, X. (eds.) *PPSN 2000*. LNCS, vol. 1917, pp. 849–858. Springer, Heidelberg (2000)
3. Angus, D.: Population-based ant colony optimisation for multi-objective function optimisation. In: Randall, M., Abbass, H.A., Wiles, J. (eds.) *ACAL 2007*. LNCS, vol. 4828, pp. 232–244. Springer, Heidelberg (2007)
4. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: *Proceedings of the IEEE International Conference on Neural Networks*, pp. 1942–1948 (1995)
5. Angus, D.: Niching for Population-based Ant Colony Optimization. In: *2nd International IEEE Conference on e-Science and Grid Computing, Workshop on Biologically-inspired Optimisation Methods for Parallel and Distributed Architectures: Algorithms, Systems and Applications* (2006)
6. Mahfoud, S.W.: Crowding and preselection revisited. In: Männer, R., Manderick, B. (eds.) *Parallel Problem Solving from Nature 2 (PPSN2)*, pp. 27–36. North-Holland, Amsterdam (1992)
7. Mahfoud, S.W.: Niching methods for genetic algorithms. PhD thesis, University of Illinois (1995)
8. Storn, R., Price, K.: Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, ICSI (1995)
9. Montgomery, J.: Alternative representations for the job shop scheduling problem in ant colony optimisation. In: Randall, M., Abbass, H.A., Wiles, J. (eds.) *ACAL 2007*. LNCS (LNAI), vol. 4828, pp. 1–12. Springer, Heidelberg (2007)

An Efficient Hybrid Algorithm for Optimization of Discrete Structures

Amitay Isaacs, Tapabrata Ray, and Warren Smith

School of Aerospace, Civil and Mechanical Engineering,
University of New South Wales, Australian Defence Force Academy, Canberra,
ACT 2600, Australia
{a.isaacs,t.ray,w.smith}@adfa.edu

Abstract. Presented in this paper is a hybrid algorithm for the design of discrete structures like trusses. The proposed algorithm called Discrete Structures Optimization (DSO) is based on the Evolutionary Structural Optimization (ESO) [1,2]. In DSO, material is removed from the structural elements based on the strain energy. DSO is a two stage process. First stage is the topology optimization where the elements of the structure with the least amount of strain energy are identified and eliminated. The second stage is the sizing optimization of the structure with optimum topology identified in first stage. For the continuous design variables a gradient based method is used and for the discrete design variables a genetic algorithm is used. The algorithm is tested on 2-D and 3-D discrete structures. DSO results show significant reduction in the number of finite element analysis (FEA) evaluations as compared to genetic algorithms using simultaneous topology and sizing optimization.

1 Introduction

The design of discrete structures such as trusses involves sizing, shape and topology optimization in the presence of applied loads and subject to stress and displacement constraints. For trusses, the sizing optimization refers to finding the optimum cross-sectional areas of the truss members keeping the connectivity of truss members fixed. In shape optimization, the location of nodes is varied. Topology optimization deals with finding the optimum connectivity of the truss members.

Topology optimization is characterized by the changes in the number of structural elements and the connectivity between those elements. Typically changes in topology are represented as discrete variables. In many of the design problems the design variables are discrete (e.g. cross-sectional areas of truss members are to be chosen from a set of available sizes). Genetic Algorithms (GAs) are heuristic, combinatorial search methods that can handle discrete variables easily and they have been found well suited for structural design problems. Initial use of GA was for the sizing optimization involving discrete variables [3,4,5,6]. GAs were later used for solving simultaneous sizing and topology optimization problems [7,8,9]; combined sizing, shape and topology optimization of trusses [10,11]. One of the

main disadvantage of using GA in structural design is the large number of finite element analysis (FEA) evaluations required.

Xie and Steven [1] introduced Evolutionary Structural Optimization (ESO) technique in 1993. It is based on the idea that the optimal structure (which is a fully stressed design) can be produced by gradually removing material from the design domain. The technique has also been applied to simultaneous sizing and topology optimization of discrete structures [2]. In ESO for trusses, the area of the truss members is reduced in very small steps (of the order of (original area)/1000 or less), if the stress in a truss member is below the target. This can lead to large number of FEA evaluations if large number of steps are required to reach the optimum structure.

The original ESO technique is applicable to problems with stress considerations only and do not consider stiffness constraints in the form of displacement constraints. To alleviate this problem Chu et al. [12] proposed the use of sensitivity numbers, which indicate the change in overall stiffness (strain energy) to remove the elements. The structural elements with the least strain energy are removed from the structure, thus achieving a faster convergence to optimum topology in the case of continuous structures. Based on the ESO approach, Taniskanen [13] has proposed a multiobjective modification to ESO with modified objective as a function of compliance volume and strain energy and also utilizes the approximations to the gradient of the objective with respect to the design variables. This method can be used only for continuous design variables. Similar methods involving material removal for truss design are presented in [14,15]. These methods are used for the truss design with discrete design variables.

Presented in this paper is an extension to ESO for discrete structures for sizing and topology optimization with discrete and continuous design variables. The proposed algorithm, Discrete Structures Optimization (DSO) is presented in Section 2. The results of DSO on 2D and 3D truss designs are discussed in Section 3. Section 4 provides the summary of the findings and conclusions.

2 Discrete Structures Optimization

Discrete Structures Optimization (DSO) is a two stage process. First stage is the topology optimization similar to ESO. In DSO, material is removed from structural elements based on the least amount of strain energy. Use of strain energy instead of stress allows for rapid material removal for the structural elements that do not carry any of the structural loads. Once the structural elements have reached the limit of material removal, they are deleted. As a result of the first stage of DSO, the optimum topology of the structure is identified. The second stage of DSO is, then, the sizing optimization using the optimum topology obtained in the first stage.

2.1 First Stage: Topology Optimization

The first stage of the DSO algorithm is as described below.

1. Start with the ground structure with all the elements initialized to the upper limit of material (e.g. maximum area possible).
2. Perform FEA to calculate stresses, displacements and strain energy.
3. If the structure is infeasible for the initial structure, there is no possible solution to the given problem. Stop.
4. If the structure is feasible, find the element with strain energy less than the threshold. Remove material from that element. Repeat from step 2.
5. If the element has reached lower limit for the material, freeze the element.
6. If the structure is infeasible, revert the material back (to the element from which it was removed) and freeze the element.
7. If all the elements are frozen, stop. If not, repeat from step 2.

Once the structural elements are frozen, they are not considered for material removal. The algorithm requires a user defined parameter - strain energy threshold. The strain energy threshold depends on the structure. Typically the strain energy for the structurally redundant element is found to be less than 1% of the total strain energy.

The material removal (in case of trusses – area reduction) in step 4 of the algorithm is achieved using polynomial mutation operator [16] as given in Eq. 1.

$$y = x + (\bar{x} - \underline{x}) \bar{\delta}$$

$$\bar{\delta} = \begin{cases} (2r)^{1/(\eta_m+1)} - 1, & \text{if } r < 0.5, \\ 1 - [2(1-r)]^{1/(\eta_m+1)}, & \text{if } r \geq 0.5. \end{cases} \quad (1)$$

For $r < 0.5$, $\bar{\delta}$ is negative and y value is smaller than x value. For $r \geq 0.5$, $\bar{\delta}$ is positive and y value is larger than x . To reduce the area x , a random number is generated between 0 and 0.5 and used as the value for r . The polynomial mutation operator constructs a density function between $\underline{x}$ and specified x peaking at x . The shape of the density function is controlled by mutation distribution index η_m . Higher the value of η_m , peakier is the density function and lower the value of η_m , flatter is the density function. The use of the mutation operator avoids the need to specify the area decrements absolutely as in the case of ESO [2].

2.2 Second Stage: Sizing Optimization

As a result of the first stage, the load bearing members are identified as the optimum topology. The second stage of DSO is the sizing optimization of the structure with the fixed topology to obtain the minimum weight structure. There are a large number of methods available for sizing optimization depending on the nature of the design variables. For continuous area variation, one can employ gradient based search methods (for faster convergence), or simplex search methods (without the need for gradient information). These methods are limited by the convergence to a local optimum. For discrete area variation methods including branch and bound methods, integer programming, etc. can be used. A discussion on various optimization methods for discrete variables can be in

found in [17]. On the other hand, one can use stochastic methods like simulated annealing (SA) or genetic algorithm (GA) to solve both the continuous and the discrete variable optimization.

In this study, a gradient based optimization is used for continuous variables. For discrete variables, Non-dominated Sorting Genetic Algorithm II (NSGA-II) [18] is used.

3 Numerical Examples

Two benchmark problems are investigated in this paper. A two-dimensional ten-member cantilever truss design and three-dimensional 25-member transmission tower design under two loading conditions [19].

Finite element analysis is performed using ANSYS (version 11.0) software. For the second stage of DSO, a gradient based optimization available as First Order Method in ANSYS is used. The specified constrained optimization problem is converted to unconstrained optimization problem using penalty function. In each iteration gradients are calculated and various steepest descent and conjugate gradient searches are performed [20]. For the discrete area variation, NSGA-II with real coding is used.

For each numerical example 40 independent runs are carried out. The parameter for mutation (distribution index) is varied with values 10, 15, 20, and 25.

3.1 A 10-Member 2D Cantilever Truss

A ground structure for six-node, ten-member 2-D cantilever truss is shown in Fig. 1. The problem is to design the minimum weight truss structure using sizing and topology optimization subject to stress and displacement constraints.

The structure is loaded with a force P of 100000 lbf at nodes 2 and 4 as shown in Fig. 1. The structure is elastic with the Young's modulus, $E = 10^7$ lb/in², and the density $\rho = 0.1$ lb/in³. The maximum allowed stress in compression or tension for all members is 25,000 lb/in². The maximum vertical displacement allowed is 2 in.

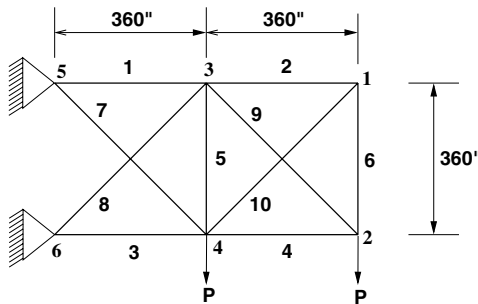


Fig. 1. Ground structure for 6-Node, 10-Member 2-D Truss

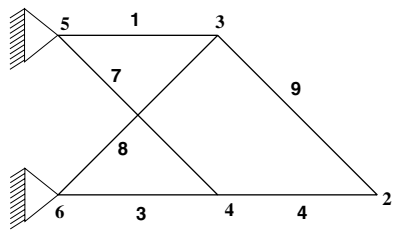


Fig. 2. Optimum structure for 6-Node, 10-Member Truss

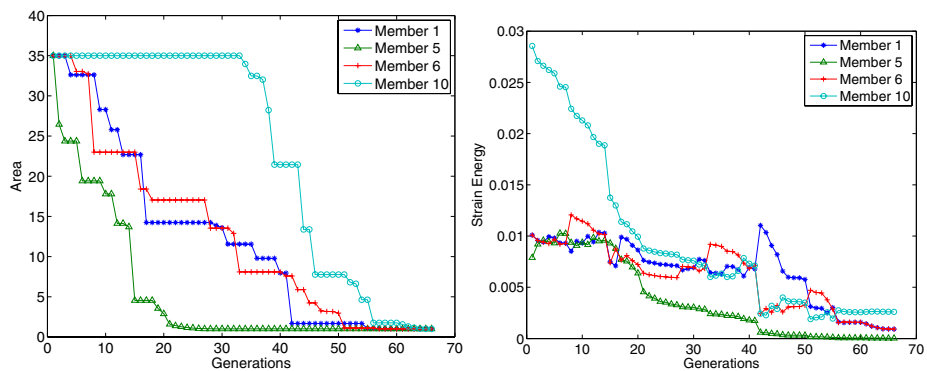


Fig. 3. Area and Strain Energy Variation with generations for 10-member 2D truss

The areas of the truss members are allowed to vary in the continuous range $[1, 35]$ in². The first stage of DSO reduces the member areas based on the lowest strain energy of the member. At the end of first stage the optimum topology is identified as shown in Fig. 2. Truss members 2, 5, 6 and 10 are deleted as they have the least strain energy. Shown in Fig. 3 is the variation of strain energy (as a percentage of total strain energy) and member areas with generations for one of the runs.

For second stage of DSO (sizing optimization), a small population of 5 individuals is created after deleting the members identified in the first stage. To ensure that the truss structures created are feasible, the area values for truss members are chosen near the upper limit using the polynomial mutation operator. (The area value specified for the mutation operator is the upper limit itself.) Each truss structure in the population undergoes sizing optimization (using gradient based first-order optimization available in ANSYS).

All the independent runs converged to the same truss design with the weight of 4961.74 lb as given in Table 1. The weight of the truss is quite close to best reported weight of 4899.15 lb by Deb and Gulati [7]. The results by Deb and Gulati are based on a population size of 220 and obtained after 225 generations corresponding to 49,500 function evaluations. Whereas, the average number of

Table 1. Results for 2D cantilever truss design for continuous and discrete area variation

Member	Continuous Area Area (in ²)	Discrete Area Area (in ²)
1	31.14	30.00
3	22.02	19.90
4	16.61	15.50
7	6.12	7.22
8	20.42	22.00
9	21.58	22.00
Weight (lb)	4961.74	4962.1
Average FEA Evaluations		
First Stage	102	54
Second Stage	437	8000

function evaluations used by DSO is 539 (102 for first stage and 437 for second stage).

Next, the member areas are chosen as set of discrete values taken from the *American Institute of Steel Construction Manual* [10]. Area = (1.62, 1.80, 1.99, 2.13, 2.38, 2.62, 2.63, 2.88, 2.93, 3.09, 3.13, 3.38, 3.47, 3.55, 3.63, 3.84, 3.87, 3.88, 4.18, 4.22, 4.49, 4.59, 5.12, 5.74, 7.22, 7.97, 11.50, 13.50, 13.90, 14.20, 15.50, 16.00, 16.90, 18.80, 19.90, 22.00, 22.90, 26.50, 30.00, 33.50). All the values are in square inches. A population of 40 is evolved over 200 generations using real coded NSGA-II. The minimum weight of the truss obtained is 4962.1 lb using average of 8054 (54 for first stage and 8000 for second stage) FEA evaluations which is the same as the reported by Kaveh and Kalatjari [8] using more than 15,000 FEA evaluations.

3.2 A 25-Member 3D Transmission Tower

A ground structure for 10-node, 25-member transmission tower is shown in Fig. 4. The tower is subject to two independent load conditions as given in Table 2. The maximum stress limit is 40,000 lb/in² in both tension and compression for all members. The maximum displacement of each joint in any direction is limited to 0.35 in.

The truss members of transmission tower are grouped based on similar characteristics. Within each group, the elements have the same cross-sectional area. The details of the seven groups are presented in Table 3.

The cross-sectional area is varied in the continuous range [0.005, 3] in². The first stage of DSO reduces the member areas (in a group) based on the least strain energy. At the end of the first stage DSO the optimum topology is identified by eliminating truss members belonging to groups 1 and 4. All the independent

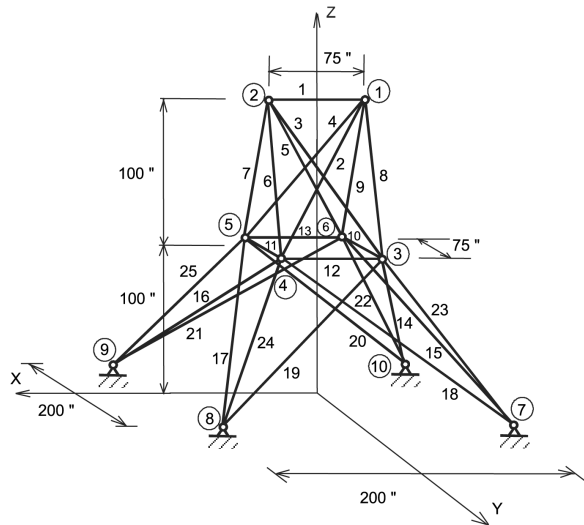


Fig. 4. Ground structure for 10-Node, 25-Member 3-D Truss

Table 2. Load cases for 25-member transmission tower

Load Case	Node #	F_x (lb)	F_y (lb)	F_z (lb)
1	1	1,000	10,000	-5,000
	2	0	10,000	-5,000
	3	500	0	0
	6	500	0	0
2	1	0	20,000	-5,000
	2	0	-20,000	-5,000

Table 3. Grouping of truss members for transmission tower

Group	Truss Elements (Connected Nodes)
1	1 (1-2)
2	2 (1-4), 3 (2-3), 4 (1-5), 5 (2-6)
3	6 (2-5), 7 (2-4), 8 (1-3), 9 (1-6)
4	10 (3-6), 11 (4-5), 12 (3-4), 13 (5-6)
5	14 (3-10), 15 (6-7), 16 (4-9), 17 (5-8)
6	18 (3-8), 19 (4-7), 20 (6-9), 21 (5-10)
7	22 (3-7), 23 (4-8), 24 (4-9), 25 (6-10)

runs converged to a single solution as reported in Table 4. The minimum weight for the tower problem reported in literature is 544.9 lb is using population size of 280 [7].

Table 4. Results for 3D transmission tower design for continuous and discrete area variation

Group	Continuous Area Area (in ²)	Discrete Area Area (in ²)
2	2.175	1.7
3	2.948	3.4
5	0.693	0.7
6	1.652	1.8
7	2.675	2.5
Weight (lb)	552.2	548.6
Average FEA Evaluations		
First Stage	72	32
Second Stage	451	8000

Next, the truss member areas are chosen from a set of discrete values. Area = (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.8, 3.0, 3.2, 3.4). All values are in square inches. Real-coded NSGA-II is used for the DSO second stage with population of 40 evolved over 200 generations. The minimum weight obtained for the tower is 548.6 lb at the cost of 8032 average number of FEA evaluations.

4 Summary and Conclusions

A hybrid algorithm based on evolutionary structural optimization (ESO) is proposed for the optimization of discrete structures. The DSO algorithm comprises of two stages. First stage is the topology optimization to identify the optimum topology (i.e. the number of structural elements and connectivity). The optimum topology is achieved by removing material from structural elements with the least amount of strain energy. The elements are not eliminated as suggested in [12] for the structures with continuous domain, but a mutation operator is used to remove material. The mutation operator can remove “large” amount of material in a single step as opposed to a very small amount of material removed in one step in ESO for discrete structures [2].

The second stage of DSO is the sizing optimization. Depending upon the nature of design variables (i.e. continuous or discrete) appropriate method for sizing is opted for. For continuous variables, gradient based methods are the fastest (using fewer function evaluations) optimization methods and hence first order gradient method available in ANSYS software is used for DSO sizing optimization. Since the gradient based methods converge to a local optimum, multiple gradient searches are utilized from different points in design space. The results show a significant reduction in the number of FEA evaluations used for the

design of trusses. For 2D cantilever truss, the average number of FEA evaluations used by DSO are 539 as compared to 49,500 FEA evaluations used by NSGA-II.

For DSO sizing optimization with discrete variables NSGA-II is used. The results obtained by DSO are comparable to the results obtained using NSGA-II for simultaneous sizing and topology, albeit using fewer FEA evaluations. It is apparent that solving the topology optimization problem separately from the sizing optimization is less expensive (in terms of number of FEA evaluations) than simultaneous topology and sizing optimization. A good choice of sizing optimization method can reduce the number of FEA evaluations even further.

References

1. Xie, Y.M., Steven, G.P.: A simple evolutionary procedure for structural optimization. *Computational Structures* 49, 885–896 (1993)
2. Steven, G.P., Querin, O., Xie, M.: Evolutionary structural optimisation (eso) for combined topology and size optimisation of discrete structures. *Computer methods in applied mechanics and engineering* 188, 743–754 (2000)
3. Rajeev, S., Krishnamoorhty, C.S.: Discrete optimization of structures using genetic algorithms. *Journal of Structural Engineering* 118(5), 1233–1250 (1992)
4. Coello, C.A.C., Rudnick, M., Christiansen, A.D.: Using genetic algorithms for optimal design of trusses. In: *Proceedings of Sixth International Conference on Tools with Artificial Intelligence*, November 6–9, 1994, pp. 88–94 (1994)
5. Wu, S.J., Chow, P.T.: Steady-state genetic algorithms for discrete optimization of trusses. *Computers & Structures* 56(6), 979–991 (1995)
6. Ghasemi, M.R., Hinton, E., Wood, R.D.: Optimization of trusses using genetic algorithms for discrete and continuous variables. *Engineering Computations* 16(3), 272–301 (1999)
7. Deb, K., Gulati, S.: Design of truss-structures for minimum weight using genetic algorithms. *Journal of Finite Element Analysis and Design* 37(5), 447–465 (2001)
8. Kaveh, A., Kalatjari, V.: Topology optimization of trusses using genetic algorithms, force method and graph theory. *International Journal For Numerical Methods In Engineering* 58, 771–791 (2003)
9. Kaveh, A., Shahrrouzi, M.: Simultaneous topology and size optimization of structures by genetic algorithm using minimal length chromosome. *Engineering Computations* 23(6), 644–674 (2006)
10. Rajeev, S., Krishnamoorhty, C.S.: Genetic algorithms-based methodologies for design optimization of trusses. *Journal of Structural Engineering* 123(3), 350–358 (1997)
11. Tang, W., Tong, L., Gu, Y.: Improved genetic algorithm for design optimization of truss structures with sizing, shape and topology variables. *International Journal For Numerical Methods In Engineering* 62, 1737–1762 (2005)
12. Chu, D.N., Xie, Y.M., Hira, A., Steven, G.P.: On various aspects of evolutionary structural optimization for problems with stiffness constraints. *Finite Elements in Analysis and Design* 24(4), 197–212 (1997)
13. Tanskanen, P.: A multiobjective and fixed elements based modification of the evolutionary structural optimization method. *Computer methods in applied mechanics and engineering* 196, 76–90 (2006)

14. Gutkowski, W., Bauer, J., Zawidzka, J.: An effective method for discrete structural optimization. *Engineering Computations* 17(4), 417–426 (2000)
15. Pyrz, M., Zawidzka, J.: Optimal discrete truss design using improved sequential and genetic algorithm. *Engineering Computations* 18(8), 1078–1090 (2001)
16. Deb, K., Goyal, M.: A combined genetic adaptive search (GeneAS) for engineering design. *Computer Science and Informatics* 26, 30–45 (1996)
17. Arora, J.S., Huang, M.W.: Methods for optimization of nonlinear problems with discrete variables: a review. *Structural Optimization* 8, 69–85 (1994)
18. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In: *Proceedings of the Parallel Problem Solving From Nature VI*, pp. 849–858 (2000)
19. Haug, E.J., Lin, C.Y.: *Introduction to optimal design*. McGraw-Hill, New York (1989)
20. ANSYS release 11.0 documentation (2007)

Evolutionary Multi-Objective Optimization for Biped Walking

Toshihiko Yanase and Hitoshi Iba

Graduate School of Frontier Sciences, The University of Tokyo, Japan
{yanase,iba}@iba.k.u-tokyo.ac.jp

Abstract. We introduce an application of Evolutionary Multi-Objective Optimization on multi-layered robot control system. Recent robot control systems consist of many simple function modules. The parameter settings for most of these modules were manually adjusted in previous research. Our goal is to develop an automatic parameter adjustment method for the robot control system. In this paper, we focused on three modules as the experiment environment: whole-body motion generator, footstep planner and path planner. At first the features of these three modules are examined. Then we discuss the trade-off relationship between the requirements of each module. Finally, we examined an application of Evolutionary Multi-Objective Optimization on this problem.

1 Introduction

The recent remarkable progress of robotics research makes advanced skills for robots to solve complex tasks. The divide-and-conquer approach is an intuitive and efficient method when we encounter complex problems. Using a divide-and-conquer approach, the multi-layered system decomposes a problem into a set of levels and each level implements a single task-achieving behavior. Many researchers employ this approach for robot control systems, dividing a complex behavior into several simple behaviors. For example, Lie et al. developed an evolutionary subsumption architecture [1], which enables for heterogeneous robots to acquire the cooperative object transferring task.

The autonomous locomotion of humanoid robots consists of following modules: global path planning using given geometrical information, local path planning based on the observation of environment, footstep planning, and whole-body motion generator. Since these modules mainly exchange information with their neighbors, we can observe that they are hierarchically arranged from the aspect of communication. The parameter settings among these modules are necessary to adapt the system to the targeting environment. The problem involves a number of conflicting objectives such as stability of the robot motion and speed of locomotion.

In this paper, we present a parameter tuning method for multi-layered robot control system by means of Evolutionary Multi-Objective Optimization (EMO). We explore the set of parameters for modules to adapt to various kind of environment. Switching these parameter sets enables effective operation of the robots.

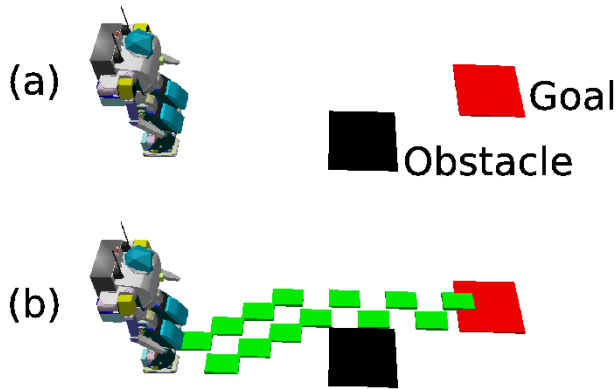


Fig. 1. Example of Footstep Planning for the humanoid HOAP-1

We developed three modules as the experiment environment: whole-body motion generator, footstep planner and path planner. In the experiment, we focused on footstep planner shown in Figure 1, which realizes the collision-free walking. The parameter settings for footstep planners were manually adjusted in previous researches [2,3]. We discuss the conflicting objectives for the optimization footstep planner, and introduce the parameter setting method by using EMO in the simulated environment.

The rest of the paper is organized as follows: Section 2 describes our robot control system, Section 3 shows experimental setting and some results, Section 4 provides the discussion of these experiments, and finally Section 5 concludes with summary and future works.

2 Multi-layered Robot Control System

2.1 System Overview

We implemented a locomotion control system which is able to find an optimal sequence of footsteps from a start location to a goal position without colliding with obstacles. The system consists of three modules, path planner, footstep planner and whole-body motion generator. The path planner generates the geometrical path for the given environmental map. Then footstep planner calculates a sequence of footsteps based on the path information. The whole-body motion generator calculates the trajectories of joint angles which realize the given sequence of footstep without losing the stability of the robot motion. By using these modules cooperatively, the locomotion planning is executed as follows:

1. Plan the entire path from the start position to the goal position by using RRT.
2. Place sub-goals for footstep planner between the path at a regular interval.

3. Plan the sequence of footsteps to the sub-goal.
4. Repeat Step 3 until the robot reaches the goal position.

In our research, we have used the HOAP-1 (Humanoid for Open Architecture Platform) robot manufactured by Fujitsu Automation. We adjusted the constraints of walking, such as the limitation of landing position and joint angles for HOAP-1. Motions were controlled by specifying the joint angles of the 20 joints of the entire body every 0.002 seconds. The characteristics of the HOAP-1 are noted below: (1) Height: 483 [mm], Weight: 5.9 [kg] (2) The internal interface between the hardware and software is available for public use. (3) For movable parts, each leg had six degrees of freedom and each arm had four degrees of freedom, for a total of 20 degrees of freedom on the left and right sides.

2.2 Path Planning

We implemented the path planning module based on the Rapidly-exploring Random Tree (RRT) algorithm [4]. RRT algorithm is a probabilistic motion planner which has found widespread use in the field of robot motion planning[5,6]. Because RRT offers almost no hypothesis in terms of the problem, we could easily apply it to various problems.

In this study, the state of the robot was expressed using 2D information (x coordinate, y coordinate), and we did not deal with other kind of information such as the rotation angle around z axis and the velocity of the robot to simplify the problem. We have to examine the collision of robots and obstacles when we use the path planning. In this experiment, the robot was approximated by the circle whose radius is 20 [cm] in order to detect the conflict between obstacle and robot.

Before generating a sequence of footsteps which satisfies the planned path, we place the sub-goal for footstep planner. The sub-goal is placed within the given distance because the computational cost for the footstep planning increases with the path length.

2.3 Footstep Planning

Footstep Planner returns an optimal sequence of footstep locations according to the plausible sets of footstep locations defined as shown in Figure 2. In this research, we employed footstep planner based on A* Search Algorithm[2,3]. The planner receives a discrete set of robot footstep locations and a 2D environment map. The search can fail in one of two ways:

- A* search tries exceeded the maximum allowable. In this experiment we limited the maximum number with 3000.
- No more valid successor node can be generated. In this case, no collision-free footstep sequence exists using the given discrete set of footstep locations.

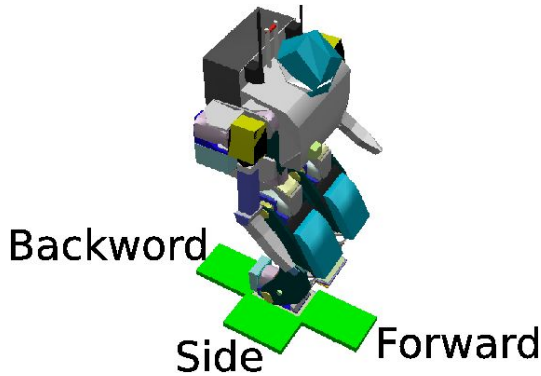


Fig. 2. Example of footsteps of the humanoid HOAP-1

2.4 Whole-Body Motion Generator

Up to now, many biped robots are successfully developed with the Zero Moment Point (ZMP) concept [7,8,9]. In this experiment, one of the ZMP based method, a Fast Generation Method of Motion Pattern that Follows Desired ZMP [8] proposed by Nishiwaki et al. was used as a whole-body motion generator. The features of this method are as follows: (1) realizing the expected feet position accurately and (2) fast enough to be used as a real-time module.

Since real robots have many performance constraints, realizability on a real robot becomes a problem. Following performance constraints are taken into consideration in this paper; (1) Existence of the solution for inverse-kinematics of legs, (2) Limitation of joint angle range, (3) Collision between links. We examined the locations of feasible states by using the simulated model of HOAP-1, and stored in the regular grid. The footstep planner referred to the regular grid to detect infeasible footstep locations.

2.5 Problems of Current Biped Walking Control System

A* Search is used for exploring a graph as shown in Figure 3, which represent the footstep transitions. The depth of the graph means number of footsteps. The spread of the graph changes depending on the number of candidate footsteps. We have to reduce the number of candidate footstep in order to calculate long footstep sequences with the same amount of searching time.

We can also observe the side-effect of discretized footstep locations. If we employ only short footsteps, the robot is able to pass through narrow paths. However, such kind of set of footsteps usually requires much computational resources for the footstep planner because short step increases the total steps. On the other hand, the set of landing positions which has only long footsteps reduces the total steps and it increases the feasible path width. In this research, we focused on two requirements, to reduce the computational resources and to minimize the feasible path width.

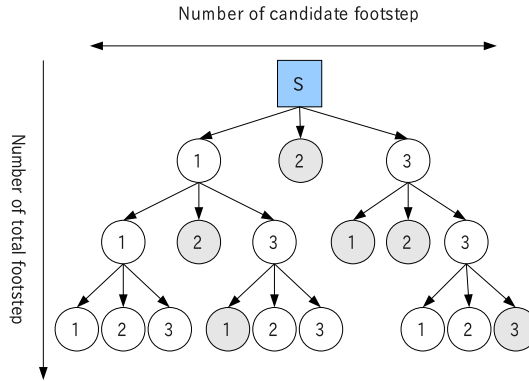


Fig. 3. Growth of search space of A* Search Algorithm. Each node represents a footstep location. Gray nodes mean infeasible locations.

3 Experiments

3.1 Experimental Setup

We used non-dominated Sorting Genetic Algorithm - II (NSGA-II) [10] which is an elitism EMO algorithm for optimization of the discrete set of landing locations. NSGA-II was widely applied for various Real-World Applications. In this experiment, a chromosome was composed of two real variable. Each variable corresponds to the length of the footstep, forward-backward and side-ward. As we mentioned in Section 2, two conflicting objectives exist. The criteria of this optimization were as follows:

- Average numbers of A* search tries
- Minimum feasible width of path

In order to evaluate these two objective functions, we created two environments: Environment A (Figure 4 left, 220cm $\times$ 220cm), Environment B (Figure 4 right, 220cm $\times$ 220cm). With Environment A, we measured average numbers of A* search tries to evaluate the computational cost. We placed the obstacles in Environment A randomly. The start position was settled in the center of the map and the goals were settled in each free space. The footstep planning was carried out from left-top to right-bottom respectively. After evaluating all the free space, we averaged the numbers of A* search tries. With Environment B, we evaluated the second objective. The task is to reach the goal located in the middle of the narrow path. We simply explored the minimum width by using hill-climbing method.

The population number was set as 100. The number of generation was set to 70. We employed the Gaussian mutation as a mutation operator, and UNDX[11] as a crossover operator. The crossover rate was 0.9, and mutation rate was 0.01. We assumed that all landing positions in the candidates can transfer to each other because we set the step cycle slow enough to satisfy the constraints.

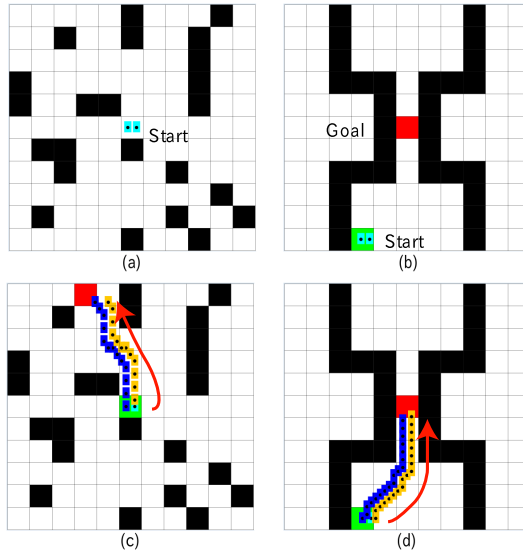


Fig. 4. Environmental maps for optimization. (a) Environment A to evaluate average numbers of A* search tries (b) Environment B to evaluate minimum feasible width of path. (c) and (d) show typical results of footstep planning.

3.2 Results

At 70th generation we observed that most of the individuals became non-dominated solutions. We also applied 10000 times Random Search as a comparison. The non-dominated solutions acquired by NSGA-II and Random Search in the typical run were plotted in Figure 5. The vertical axis means the minimum feasible width of path [mm], and the horizontal axis means the average number of A* search tries. From the Figure 5, we can observe the trade-off relationship between the two objectives. Both of the method successfully obtained the Pareto solutions, however we can observed the diversity of NSGA-II population is superior to the one of Random Search population.

We used the Ratio of Non-dominated Individuals (RNI) to compare the dominance of two populations. RNI of NSGA-II was 63% and RNI of Random Search was 37%. NSGA-II saved the number of evaluations by 30%, and successfully obtained better solutions.

3.3 Switching the Candidates for Landing Positions

In order to examine the performance of the acquired sets of landing position, we conducted another experiment. The validation environments are shown in Figure 6. There are four rooms ($2[m] \times 2[m]$) connected by the corridor ($4[m] \times 0.5[m]$). The rectangle obstacles ($0.15[m] \times 0.04[m]$) were randomly placed in the environments. We fixed the start position at the right-bottom room and

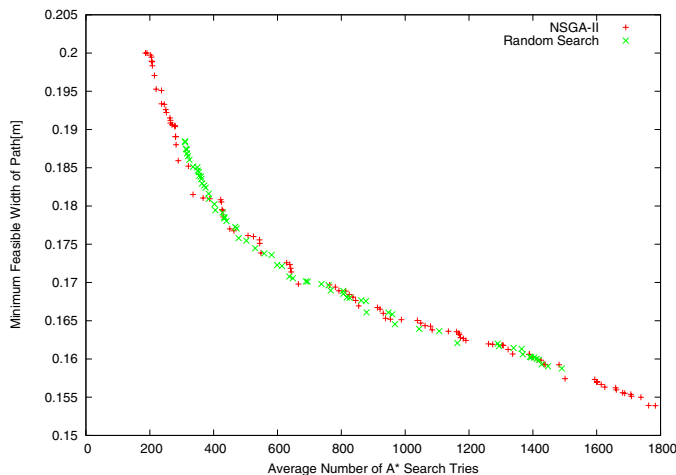


Fig. 5. Pareto-front of NSGA-II and Random Search

the goal position at the left-top room. The red circles represent the positions of the goals for the successful trials or the sub-goal when the planner failed. After generating the entire path from the initial position to the goal position by using the path planner, we added small obstacles ($0.04[m] \times 0.04[m]$). These small obstacles ignored by the path planner should be avoided by the footstep planner.

In order to reduce the number of footsteps, we switched the sets of landing positions. We used two extreme sets of landing positions, which located at both edges of the Pareto front, the set with the smallest average number of A* search tries (Set A) and the set with the smallest minimum feasible width of path (Set B). The Set A was used to reduce the total footsteps, and the Set B redeemed the failures of the plans with Set A. In this experiment, we employed a simple rule to switch the sets of landing positions as follows:

1. Execute the footstep planning by using the Set A.
2. If the planner fails to create the footsteps in step 1, apply the Set B. Otherwise, go to step 1.
3. If the sequence of footsteps is successfully calculated, go to step 1. Otherwise, the plan fails.

We compared the proposed method to the conventional method which explored the footsteps with a single fixed footstep set, the Set A or the Set B. The performance was evaluated by the total number of footsteps.

Figure 6 shows the experimental environments and a typical result of the path planning and the footstep planning. (a)-(c) in Figure 6 shows the Environment C. (d)-(e) in Figure 6 shows the Environment D. The difference between Environment C and D is the placement of the small obstacles. (a)(d) shows the footsteps

Table 1. Total number of footsteps

	planner with Set A	planner with Set B	proposed method
Environment C	-	1548	454
Environment D	398	1509	396

planned with Set A. (b)(e) shows the footsteps planned with Set B. (c)(f) shows the footsteps planned by the proposed method. The proposed method generated the short sequences of footsteps and the sequence which was able for the robot to pass through the narrow space. On the other hand, the planner with Set A failed at the middle of the map where the small obstacles and the wall formed a narrow path as shown in (a). The planner with Set B successfully passed through the narrow path, however it generated much longer sequences of footstep than the proposed method. The total number of footsteps, averaged over 5 runs are presented in Table 1. The total number of footsteps of the proposed method was about 70 % less than that of planner with Set B.

4 Discussion

In the previous section, we successfully acquired the discrete sets of landing positions for the footstep planner. By using NSGA-II, we saved the number of evaluations by 30%, and successfully obtained better solutions compared to the Random Search.

We also introduced a simple rule to switch the sets of landing positions. Using the proposed method, we obtained the 70 % shorter sequences of footsteps than that generated by the planner with Set B. We also generated the sequence which was able for the robot to pass through the narrow space where the planner with Set A failed.

In this study, we focused on the planning part of biped walking on the simulated environment. In order to apply our method to the real world, there are several difficulties to be overcome for the algorithm to function in physical robots. For example, we have to consider a feedback controller to compensate errors in motion caused by mechanical restrictions such as backlash of motors and stiffness of frames. From another point of view, we did not emphasize the mechanical constraints for the real robots, our method can be applicable to not only humanoid robots but also other biped walking agent such as 3D-CG human figures.

5 Conclusion and Future Works

We have presented a parameter setting method for a biped walking controller. At first, we introduced the robot control system and discussed the trade-off relationship of parameter settings for the footstep planning module. Next, we applied NSGA-II to acquire a set of parameters for the footstep planner. Then we proposed a simple rule to switch the sets of parameters to obtain the short

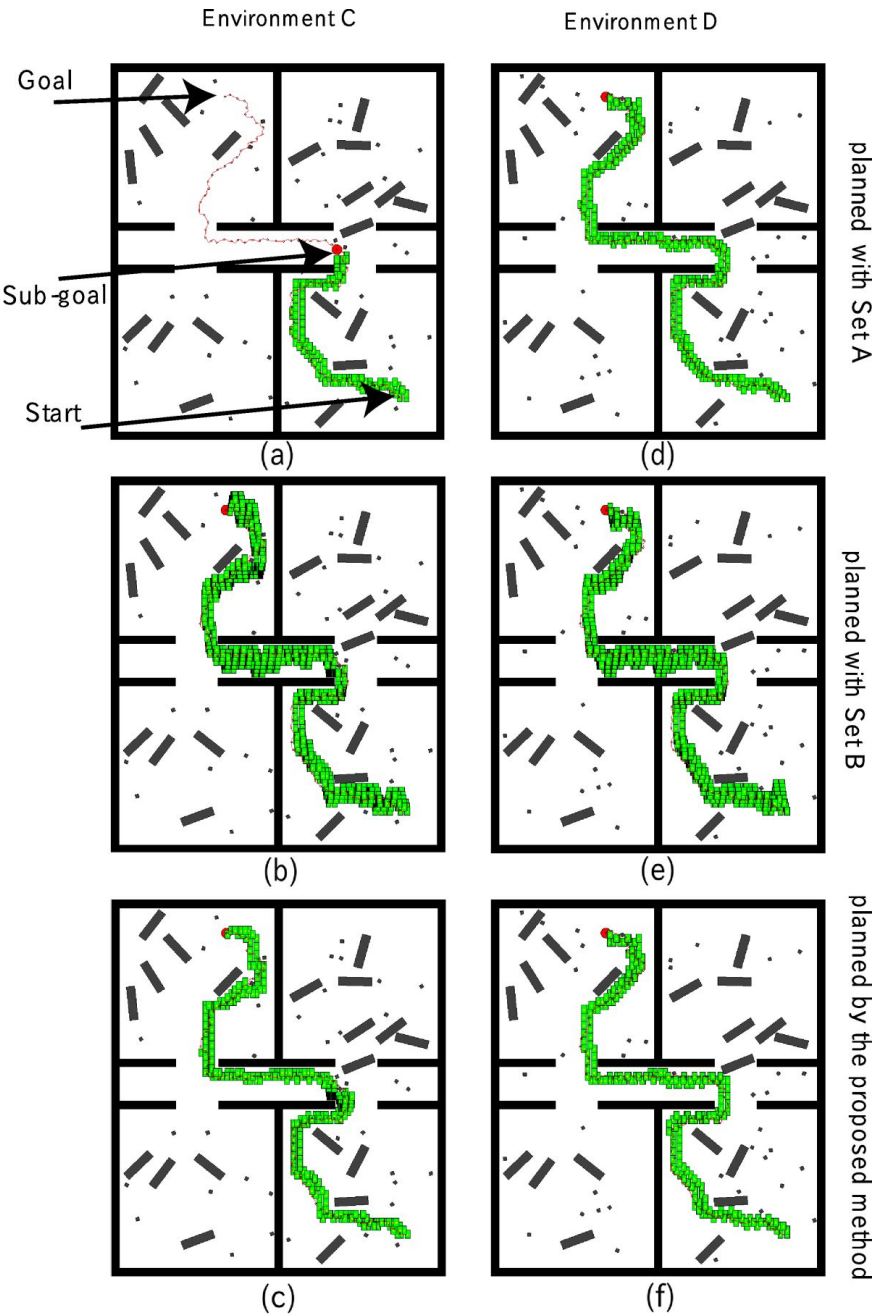


Fig. 6. Environment for validation: (a)(b)(c) show Environment C and typical results. (d)(e)(f) show Environment D and typical results.

sequence of footsteps. We also confirmed that the proposed method generated the parameters for the footstep planner which enabled the robot to pass through the narrow space.

Our current work is focusing on the translation of landing position without taking into consideration the rotation. We plan to optimize both parameters simultaneously to improve the performances of the footstep planner. Also, we intend to conduct experiments applying this system to the humanoid robot in a real environment.

References

1. Liu, H., Iba, H.: Multi-agent learning of heterogeneous robots by evolutionary subsumption. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003)*, pp. 1715–1728 (2003)
2. Kuffner, J., Kagami, S., Nishiwaki, K., Inaba, M., Inoue, H.: Online footstep planning for humanoid robots. In: *Proceedings. ICRA 2003. IEEE International Conference on Robotics and Automation*, 2003, pp. 932–937 (2003)
3. Chestnutt, J., Lau, M., Cheung, G., Kuffner, J., Hodgins, J., Kanade, T.: Footstep planning for the honda asimo humanoid. In: *Proceedings of the IEEE International Conference on Robotics and Automation* (April 2005)
4. James, J., Kuffner, J., LaVall, S.M.: Rrt-connect: An efficient approach to single-query path planning. In: *Proceedings of IEEE International Conference on Robotics and Automation, ICRA 2000* (2000)
5. Zucker, M., Kuffner, J., Branicky, M.: Multipartite rrts for rapid replanning in dynamic environments. In: *Proc. IEEE Int. Conf. Robotics and Automation* (2007)
6. Kamio, S., Iba, H.: Cooperative object transport with humanoid robots using rrt path planning and re-planning. In: *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robotics and Systems, IROS 2006* (2006)
7. Zhu, C., Tomizawa, Y., Luo, X., Kawamura, A.: Biped walking with variable zmp, frictional constraint, and inverted pendulum model. In: *Proceedings of IEEE International Conference on Robotics and Biomimetics 2004 (ROBIO 2004)*, pp. 425–430 (2004)
8. Nishiwaki, K., Kagami, S., Kuniyoshi, Y., Inaba, M., Inoue, H.: Online generation of humanoid walking motion based on a fast generation method of motion pattern that follows desired zmp. In: *Proceedings of the 2002 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, pp. 2684–2689 (2002)
9. Nishiwaki, K., Kagami, S.: High frequency walking pattern generation based on preview control of zmp. In: *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, pp. 2667–2672 (2006)
10. Deb, K., Pratab, A., Agrawal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transaction on Evolutionary Computation* 6(2), 182–197 (2002)
11. Ono, I., Kita, H., Kobayashi, S.: A real-coded genetic algorithm using the unimodal normal distribution crossover. In: *Natural Computing Series archive Advances in evolutionary computing: theory and applications*, pp. 213–237 (2003)

A Method for Assigning Men and Women with Good Affinity to Matchmaking Parties through Interactive Evolutionary Computation

Sho Kuroiwa^{1,2}, Yoshihiro Murata³, Tomoya Kitani¹,
Keiichi Yasumoto¹, and Minoru Ito¹

¹ Nara Institute of Science and Technology, 8916-5, Takayama, Ikoma, Nara, Japan
`{sho-k,t-kitani,yasumoto,ito}@is.naist.jp`

² Hopeful Monster Corporation, 8916-5, Takayama, Ikoma, Nara, Japan
`sho-k@hopefulmonster.jp`

³ Hiroshima City University, 3-4-1 Ozuka-Higashi, Asa-Minami, Hiroshima, Japan
`yosihi-m@hiroshima-cu.ac.jp`

Abstract. In this paper, we define a matchmaking party assignment problem and propose a system to solve it. The problem is to assign male and female participants to several small groups so that each group consists of the same number of men and women who have a good affinity for each other. The proposed system solves the problem based on an IEC (interactive evolutionary computation) framework, which can treat indefinable evaluation functions such as affinity between men and women by feeding back the empirically obtained values of those functions. Given each participant's attributes such as bodily characteristics, academic background, and personality, which are obtained by questionnaire in advance, the system assigns the participants to several small groups in order to maximize the number of man and woman pairs likely to begin relationships. After each groups party, the number of pairs who liked each other can be obtained as a value of the evaluation function for EC (evolutionary computation). To evaluate the system, we define the NMax Problem assuming that there would be N good affinity patterns between men and women. Through computer simulations with N from 2 to 5, we confirmed that the proposed system could find a much better group assignment than a greedy approach.

1 Introduction

Recently, the low birthrate problem has become serious in developed countries. One of the reasons for the problem is the lack of opportunities to find a marriage partner. In Japan, for this reason, several local governments have provided services that provide an opportunity for unmarried people to meet potential marriage partners. In particular, a service setting up matchmaking parties called “Gokon” is now attracting considerable attention in Japan.

Gokon is usually held by between 3 and 10 unmarried male participants and the same number of unmarried female ones, and they have a dinner party for a

couple of hours together in hope of finding a marriage partner. Compared with a traditional one-on-one marriage meeting, Gokon has the advantage that each participant can meet multiple potential partners at one time. Compared with a large matchmaking party with many participants, Gokon has the advantage that each participant has enough time to talk with each of the potential marriage partners since the number of Gokon participants is relatively small. It is important for such a small matchmaking party to assign participants to multiple small groups so that the number of man and woman pairs likely to begin relationships is maximized. However, to find a good group assignment is difficult since affinity between men and women is not yet well-understood, and the assignment problem is one of the combinatorial optimization problems.

EC (evolutionary computation) [1] is well established to solve such a combinatorial optimization problem. In particular, IEC (interactive evolutionary computation) [2], one type of EC, can treat indefinable fitness functions that cannot be evaluated in computers, such as human emotions, by feeding back the values of those functions manually. For this reason, IEC is often applied to social systems where the evaluation is difficult because of its sensuous factors [3].

In this paper, we define a Gokon (Group) Assignment Problem and propose a system to solve it. The problem is to assign many participants to several small groups taking into account good affinity patterns between male and female participants in each group. The proposed system solves the problem based on IEC. Given each participant's attributes such as bodily characteristics, academic background, and personality, which are obtained by questionnaire in advance, the system assigns the participants to multiple small groups in order to maximize the number of man and woman pairs likely to begin relationships. In the EC process, the chromosome of each individual corresponds to a concatenation of man and woman attributes representing a good affinity, and participants who have attributes similar to this pattern are assigned to the same Gokon group. After Gokon with this group assignment, EC feeds back the number of pairs likely to begin relationships, and the number is the evaluation value of the individual. Consequently, only individuals with higher evaluation values are preserved, and the system grows to obtain a better group assignment.

To evaluate the system, we define the N Max Problem, which has a function with N peaks as a benchmark test assuming that there would be N good affinities between men and women. Through computer simulations for N from 2 to 5, we confirmed that the proposed system can find a better group assignment than a greedy approach.

2 Related Work

The IEC application fields include fitting of a hearing aid [4], decision-making [5] and so on [3]. Several similar approaches have been proposed for social systems based on the IEC framework as follows.

One approach is a forum on the Web in order to obtain valuable ideas from all over the world [6]. The system that manages the forum is based on IEC.

There are many questions and their answers written in natural languages on the forum. The system can gain a knowledge database (corresponding to chromosome pool in EC) from words (corresponding to genes in EC) in each question and its answer (corresponding to chromosome). Forum participants make their answers for a question by combining only previous answers for the question (corresponding to crossover in EC) and by modifying previous ideas with their idea (corresponding to mutation in EC). Each answer is evaluated by the number of participants interested in it (corresponding to evaluation of fitness in EC). Since the interesting answers are often used to making new answers in the crossover and mutation operations, the system can gain and obtain new useful ideas easily.

Another approach is a practical teamwork course based on IEC that is part of the curriculum of engineering education at the University of Illinois [7]. Each team (chromosome) consisting of students (genes) competed to get high evaluation for the team from the members of the other teams (evaluation of fitness) during each semester (corresponding to a generation in EC). Members of each team were first assembled considering their achievements (initialization). Then, the teams were reassembled at the start of each semester by swapping members of two teams that got high evaluation (crossover) and by imitating the activity of the other teams (mutation). By this team assembling/reassembling scheme, this course could enhance its educational effect.

The IEC-based algorithms for social systems including the above approaches, generally conduct EC operations such as crossover and mutation manually. Unlike these existing applications, in this paper, we design an IEC-based algorithm for social systems which conducts EC operations semi-automatically in order to improve its efficiency.

3 Definition of the Problem

We first define the problem, which divides a large population (system participants) into small groups (Gokon) with the same number of men and women with good affinity as the “Gokon Assignment Problem”.

Input: The input of the problem is the sets of m male participants and n female participants denoted by $B = \{b_1, b_2, \dots, b_m\}$ and $G = \{g_1, g_2, \dots, g_n\}$, respectively. Each man $b_i \in B$ and each woman $g_j \in G$ have k attributes ($b_{i1}, \dots, b_{ik}$ and $g_{j1}, \dots, g_{jk}$), respectively, where each attribute represents his/her looks or personality.

Output: The Output of the problem is the assignment of B and G to h small groups (Gokon). i -th group is represented by $[B_i, G_i]$, where fractions $B_i \subseteq B$ and $G_i \subseteq G$ are assigned to a group $[B_i, G_i]$.

Objective Function: Evaluation function F , which returns the number of pairs likely to begin relationships in the group $[B_i, G_i]$, is defined by $F : B_i \times G_i \rightarrow \mathbb{N}$. Objective function is to maximize the sum of F for all groups, and is given by

$$\text{Maximize} \quad \sum_{i \in \{1, \dots, h\}} F(B_i, G_i). \quad (1)$$

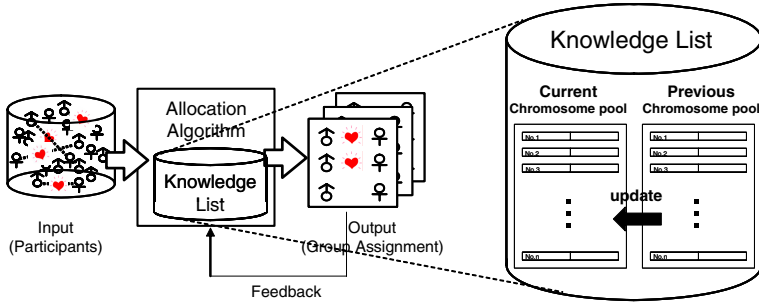


Fig. 1. Outline of the proposed system

Here, note that the evaluation function F is unknown. Thus, identifying F is also an objective of the problem.

Constraints: Conditions in which each participant is assigned to only one group at one time are given as follows: $B_i \cap B_j = \emptyset$ ($i \neq j$), $G_i \cap G_j = \emptyset$ ($i \neq j$).

The number of men and women in one group has to be the same number L . Thus, we give the following conditions: $|B_1| = |B_2| = \dots = |B_h| = L$, $|G_1| = |G_2| = \dots = |G_h| = L$.

4 An Interactive Evolutionary Algorithm for Matchmaking Party

A fitness function of the assignment problem for matchmaking party is not yet well known. Hence, we propose a system solving the problem by using interactive feedback of IEC, which can get a fitness value manually instead of defining a fitness function.

The proposed system consists of an algorithm to assign male and female participants to multiple small groups so that each group consists of the same number of men and women who have a good affinity for each other, and a *knowledge list* that holds chromosomes with high fitness values for evolutionary computation. The purpose of the system is to obtain good group assignments by improving chromosomes in the knowledge list.

The system assigns participants to multiple small groups based on each chromosome in the knowledge list. Each chromosome is coded as a concatenation of attribute sequences of a man and a woman who are likely to begin a relationship. Hereafter, we call this concatenated attribute sequence the *affinity sequence*. The first half of each chromosome shows an attribute sequence of a man, and the last half shows that of a woman (Fig. 2).

The purpose of the proposed method is to develop a system that can make as many pairs of men and women as possible begin relationships by assigning participants with attribute sequences similar to the chromosomes in the knowledge list. However, an initial knowledge list (chromosomes) is generated randomly

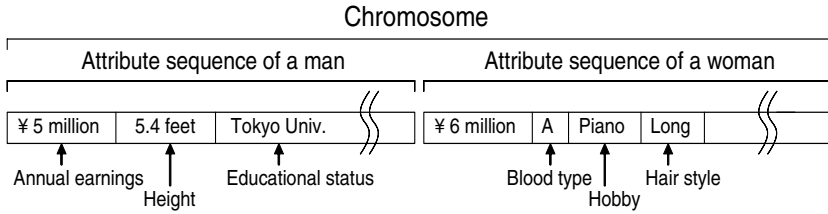


Fig. 2. Example of the chromosome representation

because good affinity patterns between men and women are unknown initially. The proposed system tries to find a set of affinity sequences representing good affinity patterns between men and women by the iterative feedback from actual matchmaking parties. After finishing actual matchmaking parties, the system feeds back the result of each party to the knowledge list, and updates the affinity sequences in the knowledge list based on interactive evolutionary computation.

Our proposed group assignment algorithm consists of 5 operators as shown below.

Algorithm
1: (1) initialization;
2: loop:
3: (2) matching;
4: (3) evaluation;
5: (4) replacement;
6: (5) selection;
7: (6) mutation;
8: if the termination condition is true then END;
9: goto loop;

Below, we describe details of the operators.

(1) Initialization: Initial knowledge list (the set of new chromosomes) is generated. The knowledge list contains the current and previous generation *chromosome pools* (current pool and previous pool) that are sets of the same number of affinity sequences as shown in Fig.1. Initially, each chromosome in the current pool is generated using random values, and the previous pool is empty. To accelerate convergence, we can give existing knowledge (i.e., the affinity sequences that are known to represent good affinity between men and women) to it.

(2) Matching: The matching operator (described below) assigns participants to groups according to chromosomes in the current pool in the following steps.

Step 2-1. For each chromosome in the current pool, find a non-assigned participant whose attribute sequence has the least distance (such as the hamming distance and the Manhattan distance) from the chromosome.

Step 2-2. Assign the participant to the group of the chromosome (see Fig. 3). If there is no group for the chromosome, make a new group for it and assign

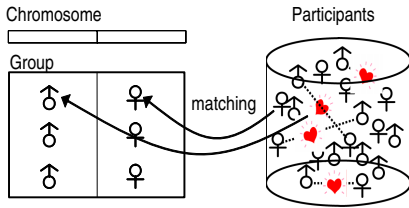


Fig. 3. Matching Operator

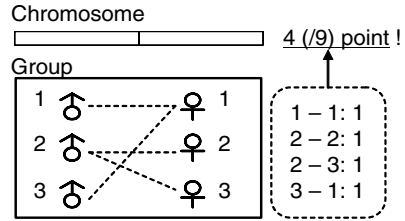


Fig. 4. Evaluation Operator

the participant. If the group is full, do not assign the participant to the group.

Step 2-3. Repeat Step 2-1 to Step 2-2 until all participants are assigned to groups or all groups of all the chromosomes are filled.

Using this operator, participants are assigned to groups according to chromosomes in the current pool (good affinity sequences). In the proposed system, participants are invited to matchmaking parties that take place by each group according to this group assignment.

(3) Evaluation: The evaluation operator gives results of the matchmaking parties as a fitness value of the corresponding chromosome. These results are obtained by the participants joining each matchmaking party through a questionnaire. After a group party, each participant answers the questionnaire about the participants of the opposite sex in the group. If a pair liked each other, we regard the pair as a good match. The system can obtain a fitness value of the chromosome corresponding to the group as the number of good matches (see Fig. 4). In the proposed method, each group consists of participants with the attribute sequence similar to the first or last half of the chromosome of the group. Therefore, it is believed that a chromosome with high fitness value leads to a lot of good matches in the group.

(4) Replacement: Replace the chromosomes in the previous pool with lower values with chromosomes in the current pool with higher fitness values.

(5) Selection: Select chromosomes for the current (next) pool, allowing the same chromosome to be selected more than once by applying the roulette selection to the updated previous pool.

(6) Mutation: Change some of attributes in each chromosome in the current pool randomly. The new chromosomes in the current pool are used to assign participants to groups for the next matchmaking parties.

5 Experiments

We conducted a computer simulation to evaluate how efficiently the proposed system can solve the Gokon Assignment Problem. For evaluation, we defined a

benchmark problem called the *NMax* problem, which is the multimodal function with N unknown peaks, where we assume each peak in the *NMax* problem corresponds to a good affinity pattern.

In the experiment, we measured the total number of good matches for the group assignments. We compared the result with two other methods: the greedy method and the oracle method. The greedy method is the conventional way to greedily search for good affinity patterns (i.e., peaks) and make group assignments. The oracle method is the way that makes group assignment with knowledge of the solutions (i.e., peaks) of the *NMax* problem.

5.1 The Benchmark Problem (*NMax* Problem)

Outline: In the EC research field, one of the famous benchmark tests is the *OneMax Problem* [8] where a bit string with all ‘1’ represents the solution. We extended this problem to have N arbitrary bit strings as solutions. We call this *NMax Problem*.

Input: Input of *NMax Problem* is a concatenation of a man’s attribute sequence $b = (b_1, b_2, \dots, b_k)$ and a woman’s attribute sequence $g = (g_1, g_2, \dots, g_l)$ such as $x = (b_1, b_2, \dots, b_k, g_1, g_2, \dots, g_{k+l}) = (x_1, x_2, \dots, x_{k+l})$ and ‘N’ peak vector $P = (p_1, \dots, p_N)$ where $p_n = (p_{n,1}, \dots, p_{n,k+l})$. In the experiment, $b = (b_1, \dots, b_k)$ and $g = (g_1, \dots, g_l)$ are given randomly like (1 0 ... 1).

In general, it is important to consider what attributes are needed and how the attributes are coded. However, in this experiment, we focus on investigating whether our proposed algorithm can effectively find solutions (i.e., N concatenations of attributes). Thus, we make a strong assumption that the *NMax Problem* can represent the existence of N good affinity patterns (attribute concatenations).

Output: Output of *NMax* is given as follows.

$$f_{NMax}(x, P) = \text{Max}_{n \in \{1, \dots, N\}} \left(\text{match}(x, p_n) \right). \quad (2)$$

Here, $\text{match}(x, p_n)$ is the function that returns the number of the same bits between x and p_n as follows.

$$\text{match}(x, p_n) = \sum_{i=1}^{k+l} \text{bitcheck}(x_i, p_{n,i}). \quad (3)$$

Here, bitcheck is defined as follows.

$$\text{bitcheck}(x_j, p_{n,j}) = \begin{cases} 1 & \text{when } x_j = p_{n,j} \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

In other words, output of *NMax Problem* is the degree of how close the input vector x is one of the peaks of the *NMax* problem. In other words, output of

the *NMax* Problem is the degree of how close the input vector x is to one of the peaks of the *NMax* problem.

Criterion to evaluate the system: We prepare the criterion “number of good matches” to check how many steady pairs form.

When inputting a group consisting of B_i and G_i as the attribute sequences of men and women, the function $F(B_i, G_i, P)$ returns the number of good matches (i.e., steady pairs). The function F is defined as follows.

$$F(B_i, G_i, P) = \sum_{b \in B_i} \sum_{g \in G_i} \text{goodmathing}(x, P), \quad (5)$$

where the function $\text{goodmathing}(x, P)$ returns whether or not x ($x \in \{bg \mid b \in B, g \in G\}$) is a good match.

$$\text{goodmathing}(x, P) = \begin{cases} 1 & \text{when } f_{NMax}(x, P) \geq \alpha(k + l), \\ 0 & \text{otherwise.} \end{cases} \quad (6)$$

Here, α is a real number representing a threshold, and we used $\alpha = 0.8$ in the experiment.

Finally, the criterion $F^*(B, G)$, the total number of good matches in all group is defined as follows.

$$F^*(B, G) = \sum_{i \in \{1, \dots, h\}} F(B_i, G_i, P). \quad (7)$$

Surely, this criterion reflects how good the group assignment strategy is.

5.2 Comparative Method

Greedy Method: The greedy method is the method where for each chromosome in the chromosome pool, the chromosome is updated to its most similar concatenation of a man and woman attribute sequence of the corresponding group, instead of “Selection” and “Mutation” operator as shown in Fig. 5. This method intends to improve the chromosomes gradually as generation progresses.

Oracle Method: The oracle method is just the way which gives solution of *NMax* Problem to the current pool in advance (Fig. 6). Each chromosome represents one of the peaks and the number of the chromosomes corresponding is the same as the number of peaks. The oracle method can find good solutions easier and faster than the proposed and greedy ones. But the oracle method is just a way to give optimal chromosomes to the chromosome pool. It does not ensure leading to an optimal group assignment. It still contains the combinatorial optimization problem.

5.3 Experimental Setup

Input data: The number of participants is 120 (60 men and 60 women), length of attribute sequence of participants is 10 bits (so the concatenation of man and

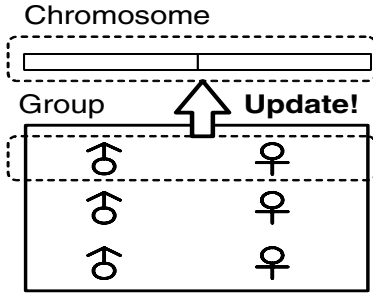


Fig. 5. Greedy Method

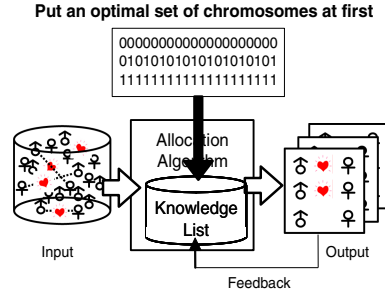


Fig. 6. Oracle Method

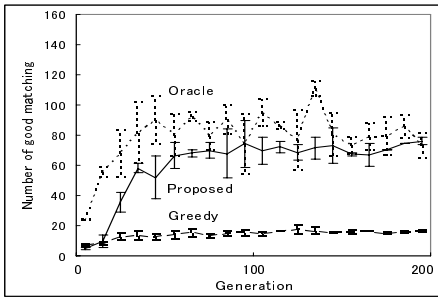


Fig. 7. 2Max Problem

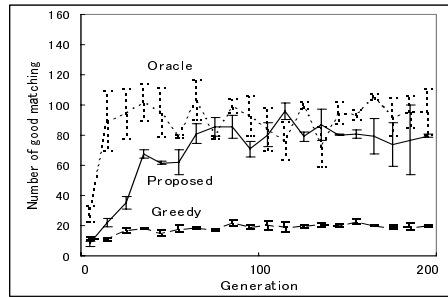


Fig. 8. 3Max Problem

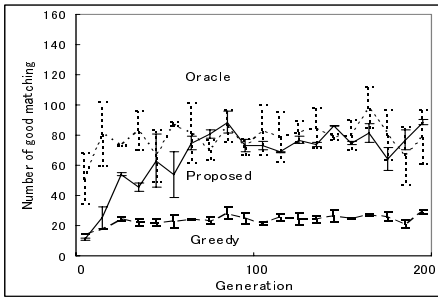


Fig. 9. 4Max Problem

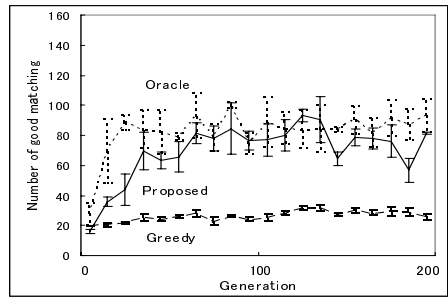


Fig. 10. 5Max Problem

woman attribute sequences is 20 bits), and group size is 6 (3 men and 3 women). The attribute sequences are initially generated by uniform random number every generation.

The EC parameters: The number of chromosomes in the chromosome pools is 20, the mutation rate is 0.1, and the selection tactics is roulette selection with 3:1 linear scaling.

5.4 Results

We show the number of good matches for N Max Problem at each generation in Figs. 7 to 10. The results of the proposed method, Greedy method, and Oracle method are represented by solid line, dotted line, and dashed line, respectively. Figs. 7, 8, 9, and 10 show the results in the cases $N = 2, 3, 4$, and 5, respectively. Each result is average value of 10 trials and error bar shows the standard deviation.

As a result, the proposed method can make good group assignments for many good matches, similar to the oracle method, and it was much better than the greedy method after the 100-th generation. For this reason, the proposed method is an effective way to make group assignments for good matches.

6 Conclusions and Future Work

In this paper, we defined the Gokon Assignment Problem, which assigns a large number of participants to small groups consisting of the same number of men and women who likely have good affinity for each other. We also proposed the Interactive Evolutionary System to solve the problem and evaluated the system using computer simulation. For the N Max problem, which is the problem to find N unknown peaks (i.e., good affinity patterns), the proposed system could find much better group assignments than the greedy method when $N = 2, 3, 4$, and 5.

As part of future research, we plan to use Evolutionary Multi-objective Clustering [9] to detect multiple peaks and Dynamic Fitness Sharing [10] to obtain diversity of solution distribution for enhancing efficiency in searching solutions.

We also plan to make group assignments with a higher degree of accuracy by estimating solutions of the N Max problem to make better assignment. For this purpose, we are now considering using regression analysis and a learning system (e.g., Support Vector Machine) at the same time.

References

1. Foster, J.A.: Evolutionary Computation. *Nature Rev. Genet.* 2, 428–436 (2001)
2. Dawkins, R.: *The Blind Watchmaker*. W.W.Norton (1986)
3. Takagi, H.: Interactive Evolutionary Computation: Fusion of the Capacities of EC Optimization and Human Evaluation. *Proc. of IEEE* 89(9), 1275–1296 (2001)
4. Takagi, H., Ohsaki, M.: Interactive Evolutionary Computation-Based Hearing-Aid Fitting. *IEEE Trans. on Evolutionary Computation* 11(3), 414–427 (2007)
5. Kalyanmoy, D., Abhishek, K.: Interactive evolutionary multi-objective optimization and decision-making using reference direction method. In: *Proc. of Genetic and Evolutionary Computation 2007*, pp. 781–788 (2007)
6. Kosorukoff, A., Goldberg, D.: Genetic algorithms for social innovation and creativity, IlliGAL Report 20001005, University of Illinois, Urbana-Champaign (2001)
7. Goldberg, D., Hall, W.B., Krussow, L., Lee, E., Walker, A.: Teamwork for a quality education: Low-cost, effective educational reform, through department-wide competition of teams, IlliGAL Report 98005, University of Illinois, Urbana-Champaign (1998)

8. Schaffer, J.D., Eshelman, L.J.: On crossover as an evolutionary viable strategy. In: Proc. Int'l Conf. on Genetic Algorithms, pp. 61–68 (1991)
9. Julia, H., Joshua, K.: Evolutionary Multiobjective Clustering. IEEE Trans. on Evolutionary Computation 11(1) (2007)
10. Della, C.A., De Stefano, C., Marcelli, A.: Where Are the Niches? Dynamic Fitness Sharing. IEEE Trans. on Evolutionary Computation 11(4), 453–465 (2007)

Author Index

- Abbass, Hussein A. 121
 Afonso, Alexandre 432
 Aguirre, Hernán 421
 Aguirre, Luis A. 121
 Allmendinger, Richard 200
 Andreae, Peter 544
 Anghinolfi, Davide 411
 Angus, Daniel 615
 Arcuri, Andrea 61

 Barbosa, Bruno H.G. 121
 Barile, Perry 299
 Berry, Marsha 259
 Boccalatte, Antonio 411
 Braga, Antônio P. 121
 Branke, Jürgen 200
 Brito, Leonardo 432
 Bui, Lam T. 121

 Cheng, Hui 370
 Cheng, Pau Chen 71
 Chua, Teck Wee 101
 Ciesielski, Vic 259, 299, 524
 Clark, John 61
 Clark, John Andrew 71
 Cullen, Jamie 514

 D'Souza, Daryl 259
 Deller, Adam 615
 Dick, Grant 452

 Elfeky, Ehab Z. 329
 Essam, Daryl L. 329

 Feng, Liang 21
 Fu, Wenlong 462

 Gao, Ya 190
 Gómez-Meneses, Pedro 229
 Goodwin, Scott 381
 Green, David G. 575
 Greenfield, Gary 51

 Halgamuge, Saman K. 319
 Halliday, David M. 11
 Hassan, Md. Rafiul 41

 Hayashi, Toshio 210
 Hendtlass, Tim 220
 Hiroyasu, Tomoyuki 339, 349
 Honda, Nobuyoshi 210
 Hossain, M. Maruf 41
 Hu, Xiao-Min 31
 Hwang, Wen-Jyi 473

 Iba, Hitoshi 635
 Iorio, Antony W. 131
 Isaacs, Amitay 401, 625
 Ishibuchi, Hisao 309
 Ito, Fuyuko 349
 Ito, Minoru 645
 Iwata, Koichiro 210

 Johnston, Mark 81, 493

 Karmakar, C.K. 41
 Khan, Gul Muhammad 11
 Kinzett, David 493
 Kirley, Michael 41, 585
 Kitani, Tomoya 645
 Kobti, Ziad 381
 Koskimies, Kai 565
 Kuroiwa, Sho 645
 Kyrki, Ville 239

 Lampinen, Jouni 239
 Lau, Henry C.W. 141
 Lehre, Per Kristian 595
 Leishman, Tania G. 575
 Li, Changhe 180, 391
 Li, Hui-Ya 473
 Li, Jinshu 289
 Li, Miqing 170
 Li, Rongxin 483
 Li, Xiaodong 131, 200, 239, 249
 Li, Yuanxiang 279
 Lim, Yow Tzu 71
 Lin, Ting-Kuan 473
 Liu, Jing 289, 462
 Lu, Jie 190

 Mäkinen, Erkki 565
 Mat Sah, Shahrul Badariah 259

- Merrick, Kathryn 1
 Miki, Mitsunori 339, 349
 Miller, Julian F. 11
 Mohemmed, Ammar W. 81
 Murata, Yoshihiro 645

 Naldi, Maurizio 151
 Neshatian, Kourosh 544
 Nishioka, Masashi 339
 Nojima, Yusuke 309

 Ortiz-García, Emilio G. 151, 360
 Ou, Chien-Min 473

 Paolucci, Massimo 411
 Paperin, Greg 575
 Paul, Topon Kumar 210
 Peng, Sheng 279
 Pérez-Bellido, Ángel M. 151, 360
 Portilla-Figueras, Antonio 151, 360
 Prothmann, Holger 111

 Rähkä, Outi 565
 Randall, Marcus 220, 229
 Ray, Tapabrata 401, 625
 Rezaei, Golriz 585
 Richter, Urban 111
 Rohatgi, Pankaj 71
 Rönkkönen, Jani 239
 Rohlfshagen, Philipp 442

 Salcedo-Sanz, Sancho 151, 360
 Santibáñez, Josefina 503
 Sarker, Ruhul A. 329
 Schmeck, Hartmut 111
 Seredynski, Franciszek 161
 Sharma, Shiven 381
 Shi, Yuan 605
 Sierra, Josefina 503
 Singh, Hemant K. 401
 Smith, Warren 401, 625
 Someya, Hiroshi 269
 Steer, Kent C.B. 319
 Switalski, Piotr 161

 Tan, Woei Wan 101
 Tanaka, Kiyoshi 421
 Tang, Cassandra X.H. 141
 Trist, Karen 299
 Trojanowski, Krzysztof 91
 Tsukamoto, Noritaka 309

 Ueno, Ken 210

 Vale, Oto 432
 Vecchiola, Christian 411
 Verma, Brijesh 534

 Wang, Peter 483
 Wang, Yu-Xuan 21
 Webb, Russell Y. 555
 Weng, Sheng-Kai 473
 Whigham, Peter A. 452
 White, David Robert 61
 Wickramasinghe, Upali 249
 Wijesinghe, Gayan 524
 Wirth, Andrew 319
 Wu, Jun 170
 Wu, Yu 279

 Xu, Xing 279

 Yanase, Toshihiko 635
 Yang, Shengxiang 180, 370, 391
 Yang, Yin-Fei 21
 Yao, Xin 61, 360, 442, 595
 Yasumoto, Keiichi 645
 Yokouchi, Hisatake 339, 349

 Zajaczkowski, Juliusz 534
 Zhan, Zhi-Hui 605
 Zhang, Guangquan 190
 Zhang, Jun 31, 605
 Zhang, Mengjie 81, 493, 544
 Zheng, Jinhua 170
 Zhong, Jing-Hui 31
 Zhong, Weicai 289, 462